Journal of Systems Architecture 129 (2022) 102567

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Reconfigurable Network-on-Chip based Convolutional Neural Network Accelerator

Arash Firuzan [a], Mehdi Modarressi [b,c,*], Midia Reshadi [d], Ahmad Khademzadeh [e]

[a] Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran
[b] School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran
[c] School of Computer Sciences, Institute for Studies in Fundamental Sciences (IPM), Tehran, Iran
[d] School of Computer Science and Statistics, Trinity College Dublin, Dublin, Ireland
[e] Iran Telecommunication Research Center, Tehran, Iran

ARTICLE INFO

Keywords:
CNN accelerator
Network-on-chip
Reconfigurable network-on-chip

ABSTRACT

Convolutional Neural Networks (CNNs) have a wide range of applications due to their superior performance in image and pattern classification. However, the performance of CNNs comes at the price of high computational load and memory bandwidth usage. Hardware acceleration has become the primary way to tackle this ever-increasing complexity of CNNs. Most of the recent accelerators arrange processing units (PEs) as a many-core accelerator architecture, with the inter-PE connections tailored to the specific dataflow of the CNN layers. The performance of such accelerators is maximized if the input feature map and filter size/dimension matches that of the underlying accelerator. However, current fixed-size accelerator structures lead to sever resource underutilization because the same structure is used to compute CNN layers of varying dimensions. In this paper, we tackle this problem by presenting RC-CNN, a reconfigurable accelerator architecture for CNNs that can adapt the structure of accelerator to the size and dataflow pattern of the running CNN layer. RC-CNN relies on a reconfigurable on-chip interconnection fabric that can organize a sub-set of accelerator's PEs as a PE set with the same size/dimension of the target CNN layer and customize the inter-PE connections for the layer's dataflow pattern. Since the area/energy overhead does not justify using a full-fledged packet-switched network in accelerators with fine-grained PEs, we use a reconfigurable network with very simple switches in order to efficiently implement the dynamic reconfiguration capability for many-core fine-grained CNN accelerators. Experimental results show that, based on the CNN size and accelerator structure, RC-CNN yields 37% higher PE utilization over a baseline design, on average. It also improves the PE utilization of the state-of-the-art CNN accelerators we selected for comparison purpose by 18%, on average. The results show that these improvements translate to 9%–41% increase in the accelerator's throughput. Further, RC-CNN reduces the network latency and energy consumption by 28% and 22%, respectively, compared to the state-of-the-art utilization-aware methods that employ packet-switched networks-on-chip.

1. Introduction

Artificial neural networks have been recently employed in a wide range of applications in our everyday life [1–3]. To cope with the ever increasing demand for accuracy in applications with growing complexity, the number of layers in artificial neural networks has increased, leading to the adoption of neural network models with a deep chain of layers, called deep neural networks. A special kind of these deep neural networks is the well-known convolutional neural networks (CNNs) [4]. Applications of CNNs are extremely widespread. These networks are customized for image processing, essential for many domains such as artificial intelligence, image and video recognition/classification, medical image analysis, pattern recognition, expert systems, natural language processing, etc. Convolutional neural networks consist of some convolution layers that each layer has a unique size in each CNN.

The convolutional layer is the core of a CNN. The convolutional layer's parameters consist of a set of filters, which have a small receptive field but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when

* Corresponding author.
E-mail address: modarressi@ut.ac.ir (M. Modarressi).

https://doi.org/10.1016/j.sysarc.2022.102567
Received 14 November 2021; Received in revised form 5 April 2022; Accepted 16 May 2022
Available online 23 May 2022

it detects some specific type of feature at some spatial position in the input.

The most impeding aspect of convolutional neural networks are the huge amount of their computation. This amount of computations is so heavy for traditional processors that they fall short to complete the computation in acceptable time. GPUs are also sometime inefficient due to their high power consumption and cost. For this reason, the semiconductor industry has moved towards the neural network accelerator chips [5–12]. These accelerator chips have a vast diversity: from small-scale embedded processing units to large cloud-scale [13–17] that can run hundred millions of neurons in real-time [18–22]. These application-specific accelerators generally consist of simple processing elements (PEs), each can perform multiple multiply-and-accumulate operations per time unit. By putting together these processing elements, an application-specific multi-core system is formed. These accelerators have a large amount of PEs connected by a network-on-chip (NoC) that distributes the data and intermediate results among cores according to the dataflow of a convolutional neural network [23–29].

Among different application-specific accelerator architectures for CNNs, the architectures based on systolic arrays often yield the best throughput and memory access reduction [30–32]. Specifically, recent accelerators device systolic array architectures based on the output-, input-, and row-stationary dataflows to maximize the reuse of fetched data. Among them, the latter, row-stationary dataflow, promise the maximum data reuse, hence better reduces the pressure on the memory bandwidth.

In such accelerators, the PEs are portioned into several so-called PE sets. Each PE set is allocated to a single convolutional layer, with layers of a CNN run one after another on the PE sets. It consists of a set of PEs and buffer banks that are connected by an interconnection network to make a systolic array. The performance of this systolic array-based architectures is highly sensitive to the match between the convolutional layer and accelerator size and dimensions.

Conventional accelerators apply the one-size-fits-all scheme, where all PE sets have the same size and these identical sets compute all convolutional layers of the CNN. Prior research, however, showed that variation in the sizes of the CNN layers limits the throughput of such conventional architectures.

If a CNN layer is smaller than the PE set size of the accelerator, then some PEs remain idle. On the other hand, if the layer larger than the PE set, the layer should be folded on the PE set and run in multiple iterations, even if there are some idle PE sets nearby. In this case, not only the execution time increases but also it is still likely to have idle PEs at the last iteration.

Therefore, for the layers that are a poor fit for the accelerator parameters, the accelerator will undergo PE underutilization problem and the existence of inefficient computational cycles, thereby increasing CNN computation time and power consumption [33,34].

Some prior work tried to work around this problem by implementing PE sets of different sizes [35] or using packet-switched network-on-chip to virtually form PEs sets [36]. The former, however, still have the underutilization problem when facing new layer sizes. The latter have a high area overhead of putting a large packet-switched router beside each PE.

To overcome the problem of varying dimensions of filters and inputs across CNN layers, this paper presents Reconfigurable NoC-Based Convolutional Neural Network Accelerator (RC-CNN), a systolic array architecture that relies on a reconfigurable NoC to dynamically form PE sets of arbitrary size. These interconnected PEs create some sets, such a way that each set can process a filter and these sets can work in parallel. To keep the NoC size and complexity low, we do not use a packet-switched NoC; rather an interconnection with simple switches is used. Each switch has the same structure as the switch boxes in FPGAs: it is programmed (configured) to connect the input and output wire segments together, hence adapting the inter-PE connections to a given dataflow. The switches make long pipelined links between communicating PEs by chaining multiple wire segments between the two end-point nodes. This capability is used to build the required inter-PE connection between the PEs that are to form a PE set. This low overhead made by the simple switches justifies putting a router along with each simple PE.

This paper takes the dataflow from the state-of-the-art row-stationary dataflow of Eyeriss [31,37,38]. RC-CNN takes the size of input CNN layers, assigns the required number (row and column) of PEs to it to form the PE set, and configures the NoC to connect the cores inside the PE set according to the row-stationary dataflow. Experimental results, as detailed in Section 5, shows that RC-CNN can considerably outperform the stat-of-the-art in terms of PE utilization and complexity reduction.

The rest of the paper is organized as follows. In Section 2, we present some preliminary concept on CNNs and introduce the state-of-the-art design, on which we have based RC-CNN. Section 3, discusses some related works. The proposed architecture for RC-CNN's is presented in Section 4 and evaluated in Section 5. Finally, Section 6 concludes the paper.

.

2. Preliminary

2.1. Convolutional Neural Networks

Artificial Neural Networks (ANNs) model the brain with a set of interconnected neurons. Neurons are the main component of the artificial neural networks. Each neuron contains a number of inputs and outputs. Synapses are connections between neurons in brain and a weight assigned with each synapse. Artificial neural networks are made up of several layers, each layer containing several neurons. . Input of a neuron is output of another neuron in previous layer.

The depth of ANNs increases rapidly every year. These Deep ANNs called Deep Neural Networks (DNNs). A special kind of these DNNs called convolutional neural networks (CNNs) which have very much impact on Deep Learning and artificial intelligence (AI). CNNs are very hot research topic because of their very wide usage from recognition to self-driving cars and AI. The main property of these networks is a huge amount of data and computations.

This paper focuses on the convolutional neural network (CNN) model. CNNs use convolution instead of general matrix multiplication which is used in artificial neural networks. Each layer in a CNN employs an operation called convolution. In CNNs a neuron only performs a simple multiply and accumulate operation and can connect to all or some part of neurons in the next layer. The outputs of one layer can be inputs of another layer. A CNN has an input layer, some hidden layers, and an output layer. Hidden layers generally have some convolutional layers. The operation of the convolution layer is equal to a neuron in the visual cortex of the brain. The layers of a CNN have neurons arranged in 3 dimensions called width, height, and depth. Where each neuron inside a convolutional layer is connected to only a small region of the layer before it, called a receptive field. CNNs are the primary way to implement deep learning-based image processing algorithms by stacking several convolutional (in the form of 2D filters) layers in front of a backend MLP-like fully-connected neural network. The filters are used to take a 2-D sliding-window convolution over an input matrix (A.K.A input feature map) to produce an output matrix (A.K.A output feature map), with the output of the preceding layer being fed to the subsequent layer as the input. Each filter extracts some features from the input feature map. Input in a CNN generally is a 2D shape or image so that each image has a height and width. The number of images may be greater than one, in such a way that called the batch size of an image. An image after passing through a convolution layer is called a feature map. Starting from the top left of the image, each filter slides across all the areas of the input image by a certain stride. At each step, the filter covers a block of the input image with the same size and

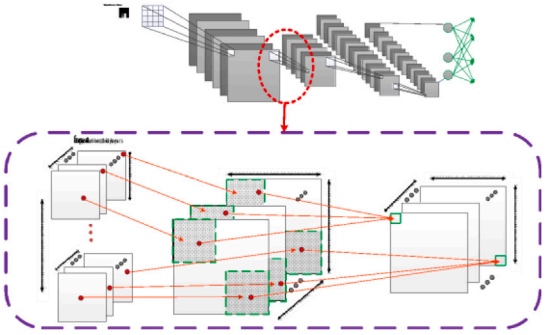

Fig. 1. Convolutional Neural Network Layer.

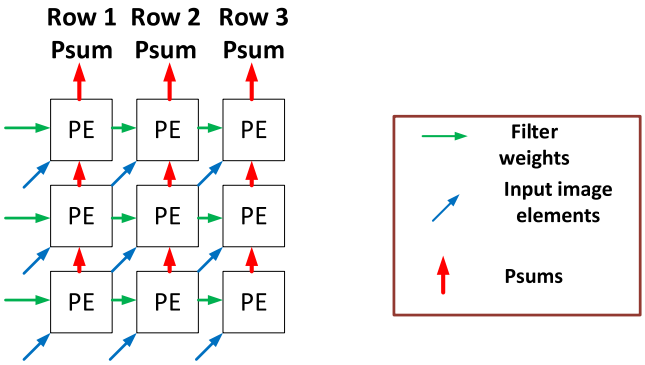

Fig. 2. Direction of filter weights movement shows in green arrows. Direction of input fmap movement shows in blue arrows. Direction of Psum movement shows in red arrows. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

calculates an element-wise dot product between the input image block and filter weights to produce an element of the output feature map.

There can be multiple (N) input channels, each containing a distinct R×C input feature map. As mentioned earlier in some papers this called the batch size of an image. In this case, there are N channels of filters, each applies to one of the input channels. The results are then summed up element-wise to make an output feature map, as illustrated in Fig. 1. Each channel has M filters to apply to the input and produces M channels of output feature maps. If input elements are from the same channel in the input image, high reuse capability can be done in PEs. In CNN accelerators, in most cases, the dimensions of the filter and the dimensions of the processing elements are not compatible, and this causes idle processing elements and a waste of resources and energy. In other words, the main problem in CNN accelerators is some PEs remain idle during CNN computations. The PE underutilization problem occurs when input activations are from the same channel. This problem is severe because most filter sizes in CNNs have small sizes. For example, AlexNet [39] has filters with size less than 27 × 27. Moreover, filter size varies in different CNN models.

One of the parameters which are used for the evaluation of CNN accelerators is efficiency and defines the number of operations per second per watt. This parameter cannot use alone because CNN accelerators cannot reach their maximum TOPS because of the PE underutilization problem. Accuracy is another parameter that is determined for a given task, the quality of the result. Throughput is an important parameter that is used to determine the amount of processed data [40]. For some applications high throughput is necessary, For example in real-time applications. Throughput express as the number of tasks that can be completed in a period of time. Latency is another important parameter. It measures the time between data arrival and result generation. Real-time CNN applications such as automatic navigation, augmented reality and robotics need low latency CNN accelerators. As mentioned earlier, the number of PEs that receive workload determines the PE utilization. In another word, PE utilization is the ratio of active PEs to all PEs on the CNN accelerator. So it is better to distribute the workload so much that all PEs become active. The task of efficient distributing of CNN layers data on PEs must be done by the interconnection network of CNN accelerator. So interconnection network or network-on-chip has a key role in a CNN accelerator. Another important parameter is dataflow. Dataflow is the order of operations and location of data storage and way of data reuse.

As mentioned earlier, multiply and accumulate is the main operation of a PE in CNN accelerators. Some weights of filter or some input activations may be zero, so the multiply operation is inefficient. These inefficient operations can be deleted. In other words, these operations can be done in fewer cycles or no cycles at all. Some extra hardware is needed to determine zero operations, thus the cost of hardware may increase.

2.2. State-of-the-art design

Convolutional neural network accelerators are a very hot research topic. Many convolutional neural network accelerators have already been introduced. One of the best state-of-the-art accelerators is called Eyeriss [31,37,38]. Eyeriss is the base architecture, which we use in this paper. Eyeriss uses 168 PEs which are arranged as a 12 × 14 PE array. There is an off-chip DRAM for storing CNN weights and input activations. To minimize input image and weight movement, Eyeriss minimizes three types of data movement such as convolutional reuse, filter reuse, and weight reuse. Eyeriss uses a CNN dataflow called Row Stationary (RS). In RS dataflow each row remains stationary in PEs. In RS dataflow, Filter rows in each line of PE array are the same, and input image rows moving diagonally in PE array. In Eyeriss architecture, one row of the filter is multiplied by one row of the input image. This multiplication is the multiplication of peer-to-peer elements. After multiplying the peer-to-peer elements, the product obtained is multiplied by the input partial sum obtained and then added by the product of the partial sum stored from the previous step to obtain the partial sum of the output of the processing element. Eyeriss minimizes off-chip DRAM accesses. It uses a network-on-chip architecture with multicast and point-to-point support. To support Multi-cast and Point-to-Point, Eyeriss uses a flat multicast NoC. Each PE in Eyeriss consists of a scratchpad, a multiply and Accumulate unit, and a control unit. The main problem of Eyeriss is the PE utilization problem especially when there are multiple PE sets in the PE array. In this case, some PEs remain Idle during CNN computations. Eyeriss benefits from compression and zero data gating techniques for energy efficiency. Fig. 2 shows filter rows reuse in a PE set in Eyeriss in green arrows. All PEs in a row have the same filter row. Fig. 2 shows data reuse of Input image in a PE set in Eyeriss. Each row of the input image is reused across PEs diagonally. Fig. 2 shows partial sums accumulate vertically across PEs. Psum of each row finally will ready on PEs on the top side of the PE set.

Eyeriss v2 is a new version of the original Eyeriss. It has a new NoC architecture in comparison to the original Eyeriss. A Hierarchical Mesh network is used for supporting a wide range of bandwidth requirements. Three types of convolutional layers can be supported by Eyeriss v2. First Conventional convolutional Layers with high data reuse, depth-wise convolutional layers with no reuse for iacts and only reuse for weights, and finally fully connected layers that have little reuse. For each data type such as iacts, Psum, and weights a separate hierarchical mesh NoC is created. Eyeriss v2 optimized for compact DNNs. When required data reuse is low then its network provides high bandwidth via unicast. In cases in which data reuse is high, it uses spatial data reuse by using Multicast or Broadcast. Eyeriss v2 uses router clusters arranged as a mesh network. Each router cluster is connected to a GLB cluster and PE cluster. Each router cluster consists

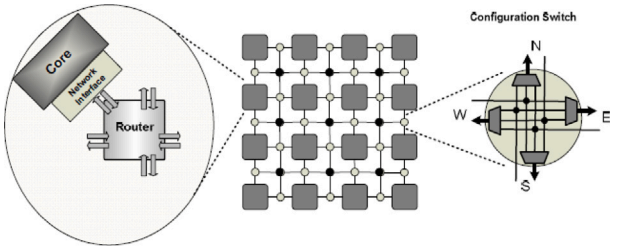

Fig. 3. Reconfigurable NoC.

of four Psum routers, three weight routers, and 3 iact routers. Each PE cluster consists of 12 PEs arranged in a 3 × 4 manner. Eyeriss v2 has a Global buffer for adding an extra level memory hierarchy. There are two control mechanisms in Eyeriss v2 similar to original Eyeriss. First is a top-level control mechanism controlling off-chip data accesses and traffic between Global Buffer and PEs and second is a control mechanism in each PE that controls the processing process in PE. Input activations read from off-chip into the GLB cluster. Partial sums always store in the GLB cluster. Eyeriss v2 can be scaled with more PEs but the performance may not be scaled due to PE utilization issues. Hierarchical Mesh NoC in Eyeriss v2 uses a circuit-switched routing with only the use of multiplexers.

2.3. Reconfigurable Network-On-Chip (NoC)

The reconfigurable topology which used in this paper is a modified version of the baseline reconfigurable topology we proposed in a previous work [41]. In the reconfigurable Network-on-Chip in [41], routers are shown with squares in Fig. 3 and they are not connected directly to each other, but through a very simple logic, which called a configuration switch. This configuration switches are shown with circles in Fig. 3 that allow changing the inter-router links dynamically. Like the switch box of FPGAS, A Configuration switch simply consists of several transistor switches, which they can be configured to create required inter-router links. These switches are configured based on the current on-chip traffic to set up permanent long connections or links between different routes. They are similar to simple 4 × 4 crossbars and are very smaller and simpler than a router because they have no arbitration, network interface, and routing logic and buffer. The crossbar is smaller than the 5 × 5 crossbar of a mesh and, the configuration switches have permanent input–output connections during application execution time unlike the internal connections of a router crossbar that may potentially change at each cycle. So the power consumption of the crossbar control line's activity is eliminated.

3. Related works

Research on convolutional neural network accelerators is very hot and interesting due to the many applications of CNNs. Interconnection used in ASIC CNN accelerators is divided into array-based [15,42], Mesh-based [26,43], custom interconnection [44,45] and finally reconfigurable interconnects [23], [24] and [46]. Interconnections in non-ASIC accelerators consist of FPGA-based accelerators such as [47], [48] and [49], GPU-based accelerators such as [50] and [51], many-core accelerators such as [52], and embedded processors such as [53] and [54]. Other interconnects and emerging technologies such as in/near-memory processing, wireless interconnections, and optical interconnects are also proposed for Neural Network accelerators. Massively parallel computing models in CNN accelerators consists of spatial architectures that use dataflow processing and temporal architectures such as SIMD architectures. The main difference between spatial CNN architectures and temporal ones is the hierarchical memory hierarchy which uses in spatial CNN accelerators. Spatial CNN accelerators have more parallelism than temporal CNN accelerators such as CPU-based architectures that use single instruction multi-data and GPU-based accelerators which use single instruction multi-thread [15,42,55]. Spatial CNN accelerators have decentralized control units. In temporal architecture, there is no direct communication between processing elements.

Multiply and accumulate is the main operation in a PE in CNN accelerators. MAC operation in PEs mainly consists of three memory reads. First, three reads for reading filter weight, fmap activation, and partial sum, and forth is a memory write for writing updated Psum. In the worst-case, all memory accesses must access off-chip DRAM which is a very energy-consuming process. So in CNN computations, the main bottleneck is memory access which must be kept as low as possible for energy-efficiency and throughput. For overcoming this problem in many previous related works for accelerating CNNs a memory hierarchy is used for decreasing off-chip memory accesses. So spatial architectures that have hierarchical memory are the best option for accelerating CNNs. For maximizing local data reuse in spatial accelerators, there are four levels of the memory hierarchy. The first is off-chip memory, the second is a Global buffer, the third is inter-PEs communication and the fourth is a scratchpad or a register file inside each PE. Mesh-based interconnects are rare in CNN accelerators. The main problem of mesh-based interconnects is the scalability problem. In other words, when the size of the PE array becomes large the mesh-based interconnects latency increases. Moreover, multi-cast is a rare data delivery pattern in CNN accelerators that cannot efficiently handle by Mesh-based interconnects. To overcome this problem, some accelerators such as CuPAN [56] proposed which benefits from Clos topology. Bus-based topologies are another approach used for CNN accelerators but massive data movement on CNN accelerators causes throughput problems in bus-based interconnects. The main problem of most CNN accelerators is the co-design of PEs and NoC which means that they only consider internal communication of one layer. In other words, dominant CNN accelerators use only certain dataflows and CNN layers which causes PE underutilization problem. Especially in cases in which other shapes of layers will map on that CNN accelerators. To overcome this problem recent CNN accelerators [46] benefit from reconfigurable interconnects which have the flexibility to math CNN accelerator with required dataflows and data communication patterns.

In the continuation of this section, we will review some new CNN accelerators that have already been proposed. Google tensor processing unit (TPU) [57] is CNN accelerator proposed by Google. This accelerator use systolic arrays which have from 4 × 4 to 128 × 128 systolic array engines with reconfigurable interconnects.

In [35] an FPGA-based CNN accelerator was proposed which can partition hardware resources for achieving better utilization of PEs. In other words, it can adapt its hardware size to the dimensions of the related CNN layer. In this paper, there are some convolutional layer processor (CLP) which CNN layers can be implemented by these CLPs. An optimization algorithm developed for finding the best multi-CLP design in this paper. Better PE Utilization than previous FPGA-based approaches is the main achievement of this paper [35].

In [34] a reconfigurable tree-based architecture was proposed for accelerating CNNs called MAERI (Multiply-Accumulate Engine with Reconfigurable Interconnect). In MAERI, communication flows are partitioned into some traffic classes named distribution, local forwarding, reduction, and collection [58,59]. MAERI can map arbitrary dataflows that exist in DNNs by using tiny switches and a reconfigurable interconnection network. MAERI consists of some multiplier switch, some adder switch, and simple switches. Each multiplier switch consists of a multiplier and 2 × 2 switch. Each adder switch consists of an adder and a 3 × 2 switch. Virtual neurons can be made by these tiny reconfigurable switches. First virtual neurons are built by architecture and then weight and input activations are distributed over PEs and in the final step output activation calculations are done. MAERI has low utilization in CNN layers with large filter sizes such as AlexNet C1

and C2, but MAERI in Newer CNNs with smaller filter sizes are more efficient. MAERI uses a chubby distribution tree with high bandwidth links near the root of the tree. These high bandwidth links on the chubby distribution tree increase cost. MAERI uses a prefetch buffer as a cache of its DRAM. A programmable controller controls the MAERI switches. MAERI uses an augmented reduction tree for data reduction and collection network. In [60] a high-performance FPGA-Based CNN accelerator was proposed. This architecture is an FPGA-based CNN accelerator architecture. In [20] an energy-efficient transferred filter-based architecture for accelerating CNNs proposed. CNN models first compressed and then accelerated in this architecture. This architecture consists of an array with a 16 × 16 PE array. In [61] a deep learning processor was proposed. This processor has a reconfigurable computing engine which consists of 4, 8 × 8 PE arrays. A tile-wise reconfiguration scheme is used in this work.

Neurocube [62] is A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. This accelerator has an architecture integrated into 3D DRAM, with a mesh-like NOC in the logic layer. SIGMA [33] is a state-of-the-art CNN accelerator that can handle CNN computations with high utilization. Its interconnection network is similar to MAERI and consists of a distribution and reduction network. As mentioned in this work the main operation in deep learning computations is general matrix-matrix multiplications (GEMMs) and the SIGMA accelerator can compute GEMMs with high performance and most utilization. Irregularity and sparsity are the main attributes of today's GEMMs. This causes poor mapping of data on CNN accelerators and the main purpose of the SIGMA architecture is to overcome this problem. In [36] a new convolution acceleration method proposed which have better PE utilization. This paper uses a packet switch method but our method which presented in this paper is a circuit-switch method which have less cost in compare to [36]. RC-CNN can adapt its interconnection network to the desired filter dimensions. This reconfiguration mechanism is based on RS dataflow on an array of PEs. Unlike other CNN accelerates that have fixed interconnection, RC-CNN can fit filter dimensions on its PEs. Furthermore RC-CNN has better performance and PE utilization in comparison to other previous CNN accelerators whether they are reconfigurable or not. RC-CNN's architectural details are provided in the next section.

4. Proposed architecture

In this section, we review the architecture of the RC-CNN and its details. The architecture of each processing element in the RC-CNN is considered as simple as the previous convolutional neural network accelerators [31], [34], [38]. As shown in Fig. 4, each processing element consists of a multiplier and a simple adder. In each step, the filter weight elements and the input image elements are multiplied and then added together with the previous result. Filter rows are read from the special memory used to store the filter and must reach the processing elements after passing through a specific path created on the RC-CNN. The same is repeated for the rows of the input image, and they must also reach the processing elements by passing through the provided path.

Given that the processing elements are the main engine performing the calculations on the artificial neural network accelerator, all the data required by the processing elements must be available on time in the input of the processing elements. This is determined by the dataflow. The dataflow determines how the required data of the processing elements are distributed in a convolutional neural network accelerator. In previous convolutional neural network architectures, dataflows were typically used that was created specifically for use in the same architecture. The importance of choosing the right dataflow or creating a new dataflow in convolutional neural network accelerators is so great that some papers only discuss creating new dataflows suitable for use in convolutional neural network accelerators.

RC-CNN uses the dataflow presented in Eyeriss architecture called Row Stationary (RS) dataflow. As mentioned in section two of this paper, in RS dataflow the filter rows are read from memory and transferred to the left edge processing elements of a set of processing elements in the RC-CNN. In this dataflow, the input image rows are also transferred to the processing elements located in the left and bottom corners. The filter rows are then moved line by line and the image rows diagonally on a set of processing elements, thus this leads to completing all the multiplication and addition operations required for convolution computations.

At the output of each processing element, a partial sum product is created, which results in the partial sums moving vertically upwards on the processing elements, and finally on the processing elements at the top edge of the set. This process is repeated until the final sum is obtained and the convolution operation is completed. For this purpose, one line of the filter and one line of the input image are mapped to each processing element. For the filter lines and the input image to be stored in the processing element, a separate memory is required inside the processing element. This memory exists in some previous architectures. This architecture also uses a small memory inside the processing elements to temporarily store the received data. In some previous architectures, this memory is referred to as scratchpad. The role of this memory (scratchpad) is to temporarily store data and filter elements or input image received from memory. As mentioned earlier, all convolutional neural network accelerators perform their calculations through processing elements. Processing elements are typically simple in most convolutional neural network accelerators. The function of these processing elements is to perform multiplication and addition operations. Here, too, the proposed convolutional neural network accelerator is built by interconnecting a large number of PEs. Connecting the processing elements in the RC-CNN is presented in a structure similar to a network-on-chip with the ability to reconfigure. RC-CNN uses the idea of a reconfigurable network-on-chip to create connections between processing elements, meaning that reconfiguration switches are used between sets of processing elements that work together on a certain filter.

As mentioned, RC-CNN consists of several reconfiguration switches and a processing element and consists of two dimensions as shown below. Reconfiguration switches are made using pass-transistors. By using pass-transistors, these switches can be configured by adjusting the gate voltage on the pass-transistors. The four input and output ports to a reconfiguration switch are named north, south, east, and west, and the reconfiguration switch allows all inputs and outputs to be connected in pairs. In other words, these switches can create the connections needed to build the paths specified by the algorithm.

4.1. The internal architecture of a PE set

Convolution neural networks are composed of several series of layers, each layer consisting of filters with specific dimensions. The accelerator presented in this paper is such that it can simultaneously implement several filters of different dimensions on its processing elements. The idea used here is that the filters that are to be run parallel on the accelerator are each assigned to a PE set. By doing this, these PE sets can each run a filter independently, in other words, these PE sets create parallel processing, which means that they can run several filters of different sizes on the architecture at the same time. Together, these two features make the RC-CNN more flexible to implement different filter sizes, unlike the previous architectures.

This reconfigurable architecture consists of several PE sets that in each PE set there are vertically, horizontally, and diagonally connections as shown in Fig. 4. As shown in Fig. 4, horizontal connections connect the processing elements in the horizontal direction and vertical connections connect the processing elements in the vertical direction. Also, there are two diagonal connections between the processing elements as shown in Fig. 4. The existence of these connections is due

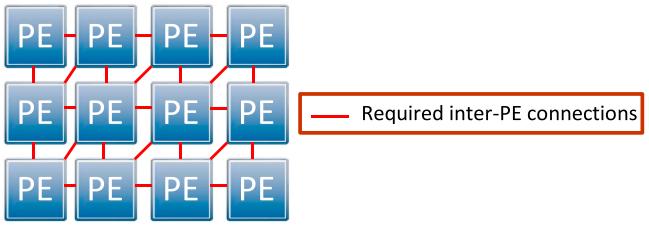

Fig. 4. Required vertical, horizontal, and diagonal connections in a PE set of the RC-CNN that are shown by red lines.

to the way the elements of the filter weights move, also the elements of the input image move, and also the movement of partial sums. As mentioned in section two in Eyeriss architecture, the elements of the filter weights move horizontally, the elements of the input image move diagonally, and the resulting partial sums move vertically. Here, too, the structure of each PE set follows the Eyeriss architecture. The elements of each PE set work together as a structure of the processing elements provided in Eyeriss architecture. As mentioned earlier, this flow of data used in Eyeriss architecture is called RS dataflow. This dataflow is used here because of its better performance than other dataflows.

Filter and input image weights are the input elements to these PE sets. Filter weights are sent through the processing elements on the left edge and inputs are also sent through the processing elements on the left and bottom edges of the PE set. Then inside each PE set according to the RS dataflow, processing operations are performed on the filter weights and inputs. This means that the elements of each filter row move horizontally and the input image rows move diagonally across the processing elements and are processed according to Fig. 2. In each step, partial sums move vertically towards the processing elements at the top edge of the PE set. In general, the number of processing elements required for each PE set to perform convolution operations is calculated according to Fig. 5. It should be noted that the number of processing elements required is not necessarily equal to the physical dimensions of the PE set. In previous convolution neural network accelerator architectures, the PE set size in most cases required to run the convolution neural network did not match the filter dimensions and at the best condition, there was little chance of fully adapting to the filter dimensions to the CNN accelerator. In other words, using the reconfiguration in RC-CNN, the size of the problem and the size of the PE set are perfectly matched.

According to the equation in Fig. 5, the number of rows required for the processing elements is equal to the number of filter rows, and the number of columns required is equal to the difference between the input rows and filter rows. Now, having the above relation, the total number of processing elements to implement each PE set will be determined by knowing the size of the filter. As an example, suppose that the dimensions of the filter are 3×3 and the input image is 5×5, in which case, according to Fig. 5, having a PE set with three rows and three columns, we will be able to perform convolution calculations on the 3×3 filter and 5×5 image.

4.2. Common filters in CNNs

One of today's new convolutional neural networks called AlexNet is shown in Fig. 6. Examination of modern convolution neural networks such as AlexNet, VGGNet[63], SqueezNet, GoogLeNet[64], and other CNNs shows that the most common filters in convolution neural networks are 2×2 filters, 3×3 filters, 5×5 filters, 7×7 filters, 9×9 filters, and 11×11 filters. Therefore, the architecture proposed for a convolutional neural network accelerator should be such that it can implement all of the above filters. An important feature of the architecture presented here is that it can implement all of the above filters.

4.3. Examination of the various configurations, including switches and processing elements

As shown in Fig. 5, the arrangement of processing elements in the number of rows and the number of distinct columns makes it possible to run different filters with different dimensions and sizes according to the equations in Fig. 5, so the number of processing elements between the reconfiguration switches can be changed. For example, as shown in Fig. 7, between each row of the reconfiguration switch, there are 4 rows of processing elements in 2 columns. In other words, there are four rows and two columns of processing elements in each PE set and also configuration switches are placed between the PE sets. To check the filters that can be implemented on this architecture, the implementation of 2×2, 3×3, 4×4, 5×5, 7×7, 9×9, and 11×11 filters have been examined. According to the results, it can be seen that we see better conditions for the implementation of 4×4, 11×11, and 3×3 filters, respectively, and the efficiency percentage is higher.

4.4. Summarize different layouts and find the optimal structure for the new architecture

Figs. 8 and 9 show different implementations on different filter sizes and the different number of columns in RC-CNN. To implement filters with different sizes, the number of architectural rows is considered as an effective parameter. Considering the standard dimensions of filters which are equal to 2×2, 3×3, 5×5, 7×7, 9×9, and 11×11, it is concluded that for the architecture to support all filters of the above sizes, the number of architectural rows must be equal to 11 or a factor of 11 to be considered. To implement the number of different columns, according to the presented architectural structure, we can consider the number of columns from two to any desired value. In this case, we will have no problem sending weights and images to the processing elements due to the presence of reconfiguration switches. As an example, an architecture with 3 columns is shown in Fig. 9 left side.

RC-CNN will have tens or hundreds of lines. But it is divided into 11 lines. To implement filters of different sizes at the same time, we must consider that the total number of rows of filters that must be applied to the accelerator at the same time must be smaller than the number of rows of architectural units, so the number of rows of architectural units should be as much as possible. The number of rows in each part of the architecture must be equal to 11. As an example, in left side of Fig. 9, an architecture has been designed and implemented that can implement 3×3 and 5×5 filters at the same time. As shown in right side of Fig. 9, in the vertical direction, each processing element is located between the configuration switches. These configuration switches allow the filter elements and the input image to enter the processing element, and after performing the multiplication and accumulation operations, they exit the processing elements and are transferred to other processing elements through the reconfiguration switches according to the dataflow. This process for calculating Convolution is repeated on the filter rows. Since the filter rows move horizontally and the input image rows move diagonally on RC-CNN, the reconfiguration switches should be arranged in such a way that the filter rows and input image can be received on the left and bottom edge processing elements on each PE set.

In general, the role of each PE set is to perform the required calculations for each filter according to the equations in Fig. 5. This feature makes RC-CNN much more flexible on the filter dimension parameter than other previous CNN accelerator architectures. Also, according to the above, unlike the common convolution neural network accelerator architectures such as Eyeriss, there is no router in RC-CNN and only reconfiguration switches are used. Configuration switches specifically include very simple hardware. They are made of only a few pass-transistors. Turning the pass-transistors on or off determines how the inputs are connected to the outputs of the configuration switches. This situation creates the desired paths on the architecture. In RC-CNN, 1024 processing elements are arranged in 32 rows and 32 columns.

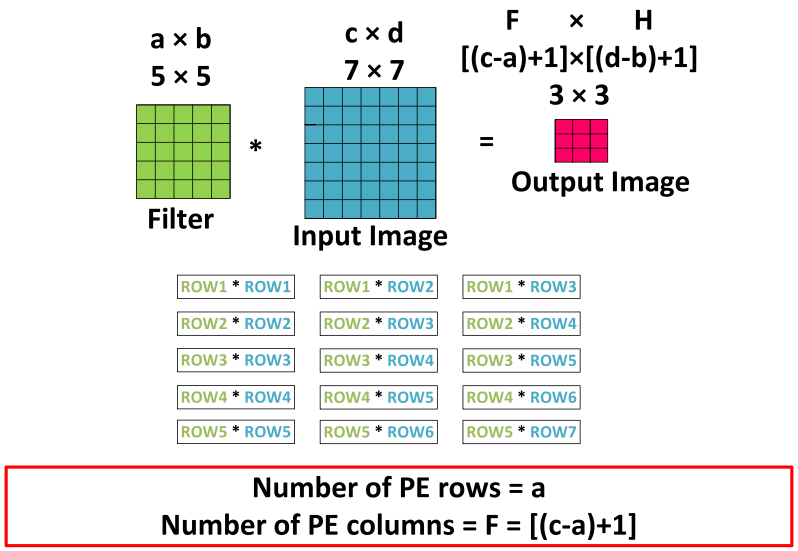

Fig. 5. Relationship between the number of required PE rows and number of PE columns inside PE set based on filter and image dimensions.

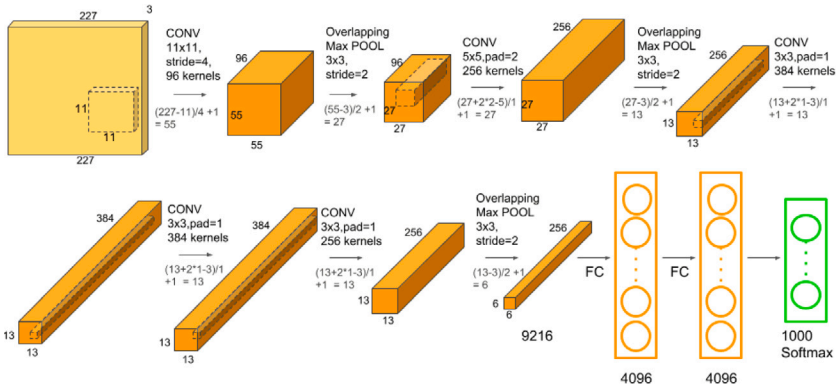

Fig. 6. AlexNet architecture [65].

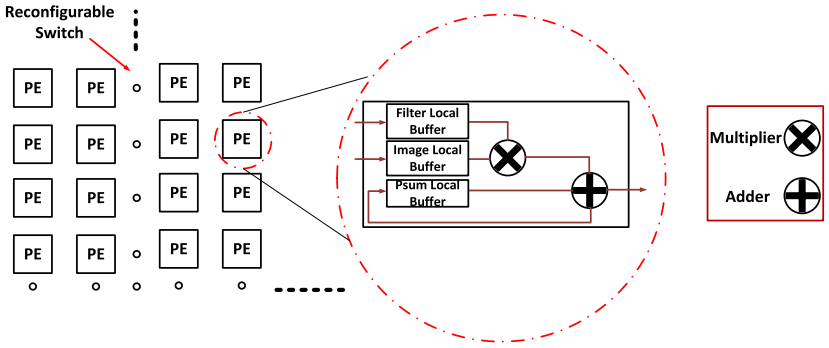

Fig. 7. RC-CNN with 4 rows and 2 columns in each PE set. For simplicity Inter-PE links not shown.

A convolutional neural network accelerator should be able to implement all the dimensions of the desired filters, due to the need for flexibility and according to the equations in Fig. 5, it should be possible to create all the common dimensions of the filters on RC-CNN. In other words, the accelerators which are presented so far have only been able to run filters of a certain size, while RC-CNN has the feature of being able to implement different filters with different dimensions. For this purpose, the filter rows should be accessible on the processing elements of the left edge of each PE set and also the input image rows should be accessible on the left edge and also the bottom edge of each PE set. To do this, the reconfiguration switches must be positioned around the processing elements so that the input image lines, as well as the filter lines, can be routed to the processing elements. As it is known, for the processing elements to have proper access to their required inputs, which are the filter rows and the input image, according to the right side of Fig. 10, a reconfiguration switch is placed in the vertical direction between each processing element so that the image rows can be accessible to the PEs at the bottom edge. Also in the horizontal direction, reconfiguration switches have been used in such a way that filter rows and image rows can be accessible for the left edge processing elements of each PE set. According to the mentioned cases, the number of columns in each PE set is considered to be two columns, so that it

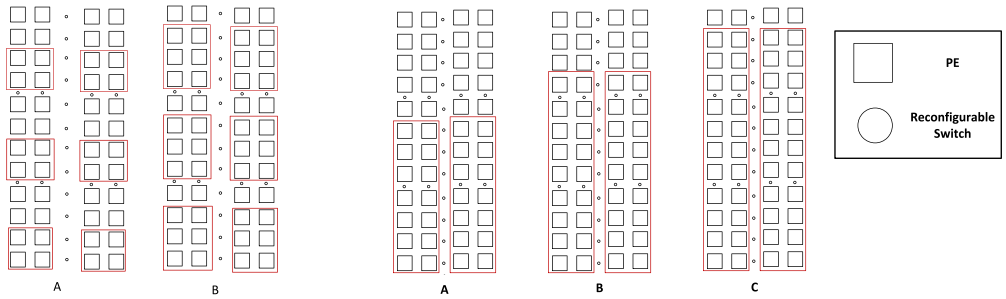

Fig. 8. (Left) (A) Implementation of 2 × 2 filters on RC-CNN, (B) implementation of 3 × 3 filters on RC-CNN. (Right) (A) Implementation of 7 × 7 filters on RC-CNN, (B) implementation of 9 × 9 filters on RC-CNN, (C) Implementation of 11 × 11 filters on RC-CNN. For simplicity, Inter-PE links are not shown on both left and right sides.

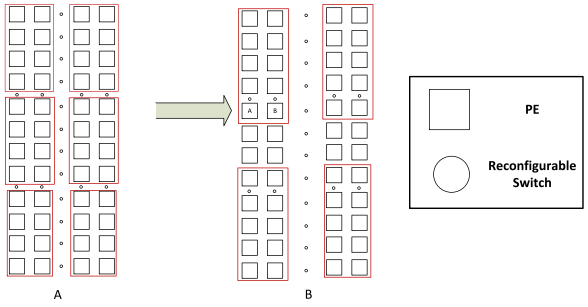

Fig. 9. (A) Implementation of 4 × 4 filters on RC-CNN, (B) Implementation of 5 × 5 filters on RC-CNN. For simplicity, Inter-PE links are not shown.

is possible to implement more filters at the same time and not need to increase the number of reconfiguration switches.

On the right side of Fig. 10. RC-CNN with 11 rows and 4 columns. Three separate memories are used to store filter elements, the input image, and the partial sums. Also, a shared-bus is used to send read data from memory to processing elements. Using separate memories will increase the speed of access to filter elements as well as input image elements, although it will increase the cost. Due to the need to achieve high-speed processing of convolution operations in convolution neural network accelerators, separate memories have been used in this architecture. Therefore, according to the above, the filter elements and the input image are stored in separate memories. Also, due to the need for data read from memory in all lines of RC-CNN, a shared-bus is used. As shown in right side of Fig. 10, filter elements and image elements are required in all rows of RC-CNN, given that the shared-bus is the best option for this. The processing elements in each cycle can perform a multiplication operation and the result is delivered to the adjacent processing element according to the dataflow.

Right side of Fig. 10, shows the purple lines of the paths created by the reconfiguration switches to deliver the input image rows to the processing elements, and the green lines also show the paths created by the reconfiguration switches to deliver the input filter rows to the processing elements. In particular, right side of Fig. 10 shows the simultaneous implementation of two 11 × 11 filters on the proposed accelerator. To better represent the PE sets created on the architecture, dotted lines are used as shown on the left side of Fig. 10. Therefore, all the processing elements inside a dotted line represent a unique PE set on the RC-CNN which a filter with dimensions according to the equations in Fig. 5 is running. The shared-bus has a controller that is used to prevent the collision of data.

When the filter elements and the input image are read from memory from cycle to cycle, they are placed on shared-bus and then transferred to the PE set via reconfiguration switches. Each memory word contains 8 bits, which are read from 64 bits of memory in each cycle and delivered to the shared-bus. The shared-bus is responsible for sending data read from memory to the reconfiguration switches on the left edge of the architecture. In general, due to the existence of multicast operations in the dataflow which is used to implement convolutional neural networks, the use of a shared-bus on the left side of the architecture between memory and processing elements is appropriate.

Inside each PE set, according to Fig. 5, convolution operations are performed on the filter elements and input image elements. The partial sum is finally present at the top edge of the processing elements on each PE set and is finally transferred to the partial sum memory via reconfiguration switches. According to the dataflow, the transfer operations of the partial sums created on the architecture start vertically from the lower processing elements on each PE set and finally, the psums are created on the upper edge processing elements on each PE set. As shown on right side of Fig. 10, these psums are transmitted through the red path created by the reconfiguration switches to the memory intended for storing the psums. The memory used in RC-CNN is DRAM.

For a better description of RC-CNN, consider the case that to implement the convolutional neural network, we need to implement two 3 × 3 filters and two 5 × 5 filters simultaneously. The left side of Fig. 11 shows an example of simultaneous implementation of two 3 × 3 filters and two 5 × 5 filters. In this case, according to the Left side of Fig. 11, first, by using an algorithm that will be discussed later in this section, the required processing elements are selected and categorized according to the equations in Fig. 5. These categories are indicated by a dashed line as shown on the left side of Fig. 11, and each category is called a PE set.

Due to the structure of the processing elements that are placed within each category, the filter lines and the input image must be accessible by the processing elements. For this purpose, first, the filter lines and the input image are read from memory and then placed on the shared-bus. The filter rows are then entered into the processing elements via the paths shown in green. The green paths are responsible for transferring data from the shared-bus to the left-hand and bottom-edge processing elements of each PE set. The two upper PE sets, indicated by the dashed line, are used to implement 3 × 3 filters simultaneously, and the two bottom PE sets are used to implement 5 × 5 filters. After the filter rows and input image data reach the left and bottom edge processing elements, they reach the other processing elements within the PE set through the connections within each PE set as shown in Fig. 5. Simultaneously with the multiplication and addition calculations required to perform convolution operations on the processing elements, the resulting psums move upwards and after reaching the output of the upper edge processing elements through the red paths formed by the reconfiguration switches enter the resulting memory of psums. This operation is repeated by reading the new lines of the filter and the input image until the convolution operation is performed on all lines of the image and filter.

The right side of Fig. 11 shows another example of RC-CNN. As can be seen on the Right side of Fig. 11, RC-CNN implements two 11 × 11 filters simultaneously. For this purpose, two PE sets are displayed with a dashed line, each PE set containing 11 rows and 2 columns of processing elements. According to the structure of the 11 × 11 filter and according to the equations in Fig. 5, the structure presented in this

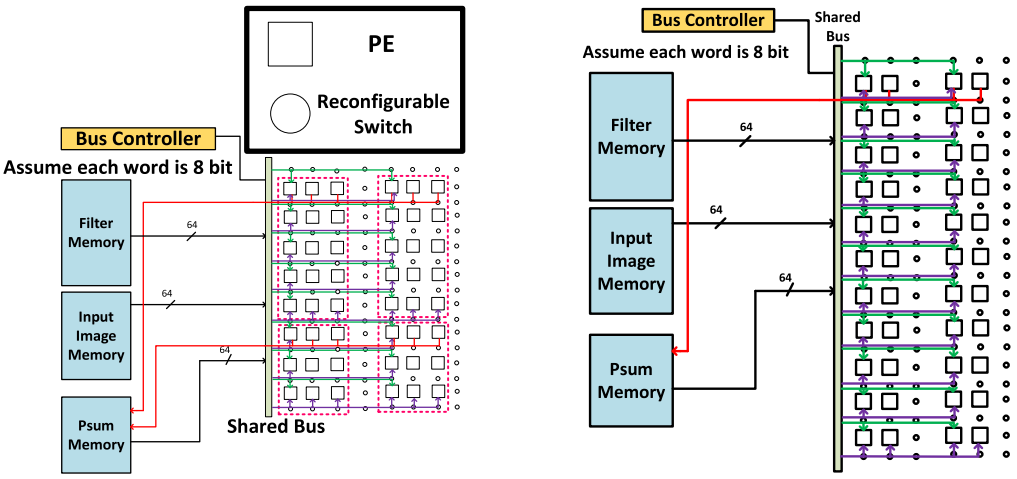

Fig. 10. (Left) Proposed architecture with 3 columns. (Right) RC-CNN with 11 rows and 4 columns. For simplicity Inter-PE links not shown in both left and right sides. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

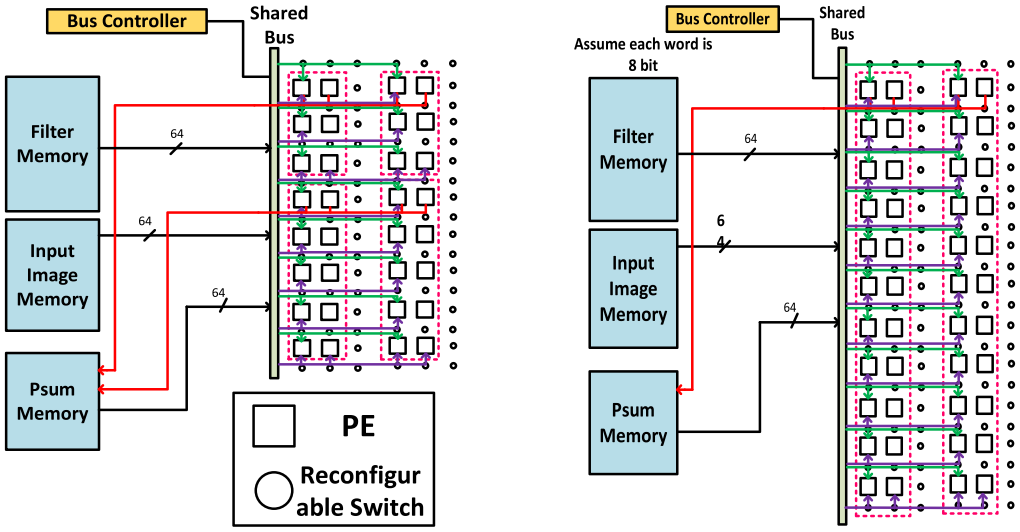

Fig. 11. (Left) An example of simultaneous implementation of two 3 × 3 filters and two 5 × 5 filters. (Right) RC-CNN implements two 11 × 11 filters simultaneously. For simplicity Inter-PE links not shown in both left and right sides.

architecture can adapt itself to the dimensions of the desired filter so that the maximum possible use of processing elements is done.

One way to increase the speed of access to rows of the input image is to use multiple memories to store the input image instead of one memory. As shown in Fig. 12, instead of adopting a single memory module, several memories are used for the input image. Also, instead of using one memory, two memories are used to store the filter. Although this will increase the cost, the increase can be mitigated by the faster access to filter elements and input image in convolutional neural network accelerators. The number of filters that can be implemented simultaneously in Fig. 12 are the two 11 × 11 filters marked by the paths created in green and purple.

4.5. Algorithm for selecting processing elements in a PE set

As mentioned earlier, to select the processing elements so that they match the dimensions of the required filters, an algorithm is proposed that can create the desired PE set by taking the dimensions of the filters as input. In this way, after creating the required PE set that can calculate the desired filter, you can configure the desired filter on the PE set by configuring configuration switches. For this purpose, after receiving the required dimensions of the filters and according to the equations provided in Fig. 5, the algorithm calculates the required number of rows and columns and then according to the obtained parameters, categorizes the processing elements according to the dimensions of the filters. In this algorithm, the classification of processing elements starts from the top of the architecture and continues downwards. If the dimensions of the filter are 3 × 3, three rows of filters are selected. Also, if the dimensions of the filter are 5 × 5, then 5 rows are selected from the processing elements in the architecture.

5. Experimental results

In this section, we compare RC-CNN with a baseline and some state-of-the-art accelerator designs in terms of utilization and execution speed. We first evaluate the utilization for each CNN layer, in order to study the behavior of RC-CNN for different filter sizes. Then, the latency, throughput, and utilization for the entire CNN benchmarks are tested. Afterwards, we compare RC-CNN, which is a circuit-switched network in nature, to a state-of-the-art packet-switched network that

ALGORITHM 1: PE Selection Algorithm

```
Start:
 Select PEs
 {
        Give filter dimensions as the input
      Define N as the number of columns in each PE set
        Go to PEs at the upper side of architecture
        If (filter size = 3×3)
            Then select a 3×N PE set from the top side of the architecture
        Else if (filter size = 5×5)
            Then select a 5×N PE set from the top side of the architecture
        Else if (filter size = 7×7)
            Then select a 7×N PE set from the top side of the architecture
        Else if (filter size= 11×11)
            Then select an 11×N PE set from the top side of the architecture
        Endif
 }
 If there are empty rows in the architecture
 Top = bottom end of the last PE set created
 Go to start
 Connect weights and inputs from memory to PEs
 {
        1. Read all Rows of the weight matrix And send them to the right side of the PE set.
        2. Configure reconfigurable switches in each PE set so that establish diagonal connections
        between PEs inside PE set.
        3. Send weight matrix row horizontally until reach all PEs in related row of PE set.
        4. Read all Rows of the input image and send them to the right and bottom corner.
        5. Configure reconfigurable switches in each row of PE set so that West port connect to
        East port.
        6. Send input image rows diagonally by configuring reconfigurable switches until reach
        PEs on                                  right and top corner of the PE set.
 }
      Deliver Partial sums from the top of each column to memory
      Connect top PE of each column to the memory
 End
```

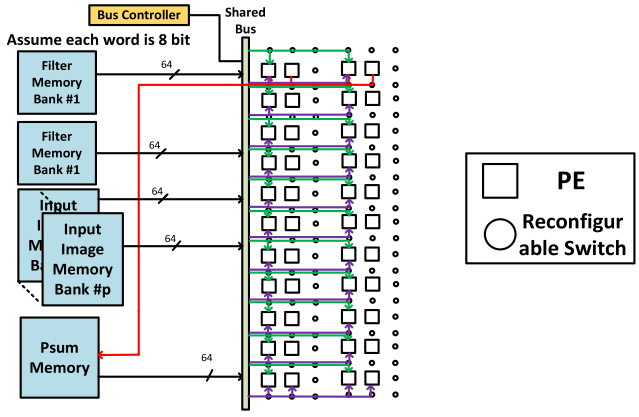

Fig. 12. Multiple memory banks in RC-CNN. For simplicity, Inter-PE links are not shown. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

is presented in a recent work with the same goal as RC-CNN, i.e. increasing resource utilization for CNNs. Finally, the cost of impending RC-CNN is evaluated.

5.1. Benchmarks

We use some widely-used CNN architectures, namely AlexNet, VGGNet, SqueezNet, and GoogLeNet as benchmark. *T* The benchmarks are trained by TensorFlow. The specification of each benchmark is outlined in Table 1. AlexNet is a CNN that consists of 8 layers and each layer consist of various filter sizes such as 11 × 11, 5 × 5 and 3 × 3. GoogLeNet is another widely-used CNN that is developed by Google to run large AI applications and it consists of 22 Layers. In overall, GoogLeNet has 1024 3 × 3 filters, 256 5 × 5 filters, and 96 filters of size 11 × 11. VGGNet is another CNN with 3 × 3 filters. Also, SqueezNet is high-accuracy but complex CNN that consists of 3 × 3 filters.

Table 1
CNN benchmarks.

Benchmark CNN	Image size	Number of layers	Filter sizes
AlexNet [39]	227 × 227	8	11 × 11, 5 × 5, 3 × 3
VGGNet (VGG16) [63]	224 × 224	16	11 × 11, 5 × 5, 3 × 3
SqueezNet [66]	224 × 224	10	3 × 3
GoogLeNet [64]	224 × 224	22	3 × 3, 5 × 5

5.2. Evaluation metrics

To compare the RC-CNN with other state-of-the-art accelerators, we used some important metrics that reflect how efficient an accelerator executes CNNs [40]. Evaluation parameters used in this paper are PE utilization, accelerator throughput, benchmark execution latency, and accelerator area. According to [40], utilization is defined as the ratio of the number of active PEs to all PEs on a CNN accelerator when running a benchmark. Utilization shows the ability to distribute the workload to PEs and fit the benchmarks into the accelerator structure. Throughput shows processed data in a period of time. The higher throughput means higher processing bandwidth. As mentioned earlier, idle PEs degrade the throughput of a CNN accelerator. Latency for processing a CNN is another parameter that has been evaluated by simulation and indicates how fast a benchmark can be run.

5.3. Methodology

To achieve a fair comparison, we compare the RC-CNN with a baseline CNN accelerator and two state-of-the-art CNN accelerators, Eyeriss [31,37] (introduced in Section 2.2) and Multi-CLP [35] (introduced in Section 3). All accelerators have 256 PEs. In RC-CNN the PEs are arranged in 64 rows and 4 columns, as shown in the right side of Fig. 11 (note that Fig. 11 shows only 11 columns for the sake of simplicity).

Baseline architecture consists of 256 PEs, arranged as 16 PE sets of size 4 × 4. The baseline's PE sets are connected by a mesh network. Larger filter sizes are serialized on the PE sets. For example, a 5 × 5 filter will run on a PE set in two iterations and an 11 × 11 filter will run on a PE set in four iterations, with three rows of the filters are mapped to the PE set at each iteration. This way, the partial sums will be stored locally and finally, partial sums will sum together to produce the final result.

We model RC-CNN and the other considered accelerators by developing a C++custom-built mapping tool and a cycle-accurate simulator. Fig. 13 demonstrates the tool chain used for the evaluation process. The convolutional neural networks are trained by Tensorflow. To run each CNN benchmark, a configuration generation tool takes the required TensorFlows output, that is the neural network hyperparameters (topology, number of layers, and filter sizes) and also the accelerator parameters (number of PEs and PE set size) and configures RC-CNN and schedules the CNN layers to run on it. The tool implements our proposed algorithm, which is explained in Section 4. Then a cycle-accurate custom-built simulator is used to take the RC-CNN configuration and the execution schedule of a target CNN from the previous tool (configuration generation tool) and calculate latency, throughput, and utilization results.

The baseline and the other considered accelerators are also implemented by a cycle-accurate C++ code.

In all configurations, input feature maps and weights are quantized to 8-bit fixed-point numbers. Prior works show that in this bit-width, all CNNs can still preserve acceptable accuracy level [1].

5.4. Comparison results

Utilization comparison. Tables 2 to 7 show a detailed comparison of layer-based PE utilization between the proposed architecture and the other considered state-of-the-art CNN accelerators.

Table 2
PE utilization under the AlexNet layers.

AlexNet layer	Filter size	Baseline	Eyeriss	Multi-CLP	RC-CNN
CNV1	11 × 11	60%	80%	87%	94%
CNV2	5 × 5	57%	86%	89%	95%
CNV3	3 × 3	58%	88%	92%	96%
CNV4–5	3 × 3	59%	87%	88%	98%
Average		58.5%	85.25%	89%	95.75%

Table 3
PE utilization under the GoogLeNet layers.

GoogLeNet layers	Filter size	Baseline	Eyeriss	Multi-CLP	RC-CNN
CNV1	7 × 7	57%	81%	83%	92%
CNV2	3 × 3	47%	87%	88%	95%
CNV3	3 × 3	58%	86.5%	87%	93%
CNV4	3 × 3	59%	89%	89%	94%
CNV58	3 × 3	60%	90%	90%	96%
CNV59	3 × 3	61%	91%	93%	97%
Average		57%	87%	88%	95%

Table 2 shows PE utilization under AlexNet, which consists of five convolution layers. According to Table 2, the average PE utilization in RC-CNN is 95.75%, with the best utilization (more than 98%) belonging to layer four (CNV4). This high performance occurs because of CNV4 layer has small 3 × 3 filters which can be easily mapped on RC-CNN. The lowest utilization is 94% for layer CNV1, mainly due to its large 11 × 11 filters. Even for this layer, the reconfiguration capability of RC-CNN can handle this filter size with moderate utilization loss. In all layers, RC-CNN outperform the other considered methods, since it can dynamically adapt the PE set size to the running layer.

Table 3 shows PE utilization on GoogLeNet, which consists of 59 convolution layers. The first layer of GoogLeNet has 7 × 7 filters, but other layers have 3 × 3 filters. Like some prior work, simulation results for GoogLeNet are presented on some selected layers, which are layers 1 to 4 and 58 to 59. According to Table 3, the average PE utilization in RC-CNN when running GoogLeNet is 95% and best utilization is 97% on layer CNV59. This occurs because of CNV59 layer has small 3 × 3 filters, which is easier to handle by RC-CNN, and the worst utilization is 92% occurred on layer CNV1 because of its large 7 × 7 filters. As shown in Table 3, the baseline architecture due to use mesh topology in its architecture have less PE utilization, while Eyeriss have better PE utilization due to its multicast-supported architecture. Multi-CLP in CNV1-3 layers have better PE utilization in compare to Eyeriss due to its parallel CLPs and have same PE utilization in CNV4 and CNV58-59. Another behavior demonstrated in the table is different utilizations for layers CONV2 to CNV59, which have filters of the same size. The reason is that the PE utilization does not depend only on the dimensions of the filter, but also on the number of filters and channels of the layer. Larger layers have more filters to keep the PEs busy, thereby increase resource utilization. So the difference in utilizations of Table 3 is not just because of the dimensions of the filter, but also because the layers become larger in deeper CNN layers.

Table 4 shows PE utilization in SqueezNet Benchmark. According to Table 4, for the first convolution layer of SqueezNet, which contains of 7 × 7 filters, RC-CNN exhibits 9% to 44% improvement over the other state-of-the-art CNN accelerator architectures. RC-CNN performs slightly better on the rest layers, with 3 × 3 filters, and keeps roughly the same improvement over the rivals.

Table 5 shows PE utilization on VGGNet (VGG16) CNN. According to Table 5 first convolution layer of VGGNet, which consists of 3 × 3 filters, gives 14% to 36% improvement over the competitors. as observed for the previous benchmarks, next layers, which larger sizes, show slightly better utilization. The average improvement over the baseline, Eyeriss, and Multi-CLP is 37%, 16%, and 12%, respectively.

Average PE utilization comparison. After the layer-wise analysis of Tables 2 to 5, Table 6 shows a comparison between RC-CNN and

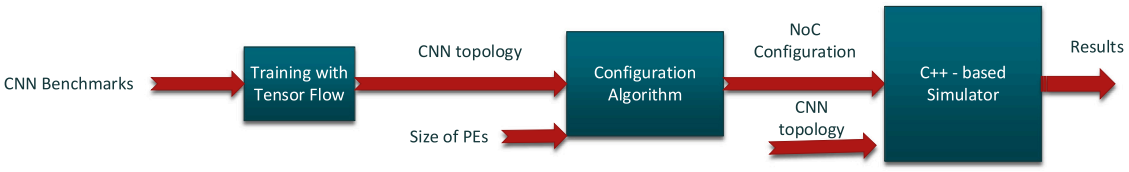

Fig. 13. Simulation process and tool chain.

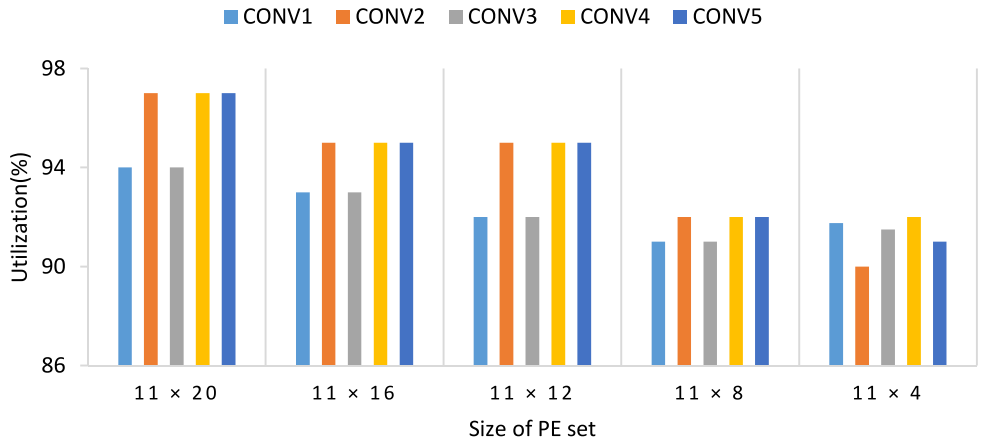

Fig. 14. PE utilization of RC-CNN on various accelerator sizes (number of PEs) under AlexNet layers.

Table 4
PE utilization under the SqueezNet layers.

SqueezNet layers	Filter size	Baseline	Eyeriss	Multi-CLP [35]	RC-CNN
CNV1	7 × 7	54%	88%	89%	98%
CNV2	3 × 3	56%	85%	86%	99%
CNV3	3 × 3	57%	87%	88%	100%
CNV4	3 × 3	52%	89%	90%	100%
Avg		54.75%	87.25%	88%	99.25%

Table 5
PE utilization under the VGGNet layers.

VGGNet(VGG16) layers	Filter size	Baseline	Eyeriss	Multi-CLP [35]	RC-CNN
CNV1	3 × 3	61%	81%	83%	97%
CNV2	3 × 3	58%	89%	91%	98%
CNV3	3 × 3	63%	86%	87%	100%
CNV4	3 × 3	63%	84%	88%	100%

Table 6
Average PE utilization of the CNN benchmarks (for the entire CNN).

CNN name	Baseline	Eyeriss	Multi-CLP [35]	RC-CNN
AlexNet	59.5%	84.5%	86%	97.5%
VGGNet	55%	87%	89%	99%
SqueezNet	60%	83.5%	87%	99%
GoogLeNet	51%	85.5%	88%	95%

other previous state-of-the-art architectures on average PE utilization when running the entire CNNs. Table 6 shows RC-CNN yields up to 86% improvement in PE utilization over the baseline (for GoogleNet). Compared to the state-of-the-art accelerators, the maximum improvement is 19% (13% on average).

Execution time comparison. Table 7 shows a comparison between the execution time of CNN benchmarks on RC-CNN and other considered state-of-the-art architectures. The execution time of the benchmarks is computed in terms of the execution cycles. Since we do not change the PEs' internal architecture, all accelerators are assumed to work in the same clock frequency and each MAC operation is assumed to be completed in a single cycle.

Table 7
Execution time for running CNN benchmarks (K cycles).

	Baseline	Eyeriss	Multi-CLP [35]	RC-CNN
AlexNet	215	186	160	146
VGGNet	415	350	337	320
SqueezNet	8924	7494	6802	5494
GoogLeNet	9870	6780	5380	4780

Table 8
Comparison of throughput in CNN accelerator architectures.

	Baseline	Eyeriss	Multi-CLP [35]	RC-CNN
Throughput	0.59	0.88	0.91	1

Table 7 shows that RC-CNN has less execution time compared to other previous state-of-the-art architectures. The improvement comes from the higher resource utilization of RC-CNN, by which more PEs are deployed to run the CNN layers. The average improvement over the baseline is 37%. Compared to the stat-of-the-art accelerators, the average improvement is 16%. The trend of the results presented in Table 7 is correlated with the utilization results presented in Table 6.

Throughput Comparison. Table 8 shows a comparison between throughput in RC-CNN and other state-of-art-architectures in terms of throughput averaged across all CNNs. As shown in Table 8, RC-CNN has 9% to 41% improvement in throughput parameter compared with the considered accelerators. The main source of this improvement is the higher execution speed that in turn, is a result of higher resource utilization.

Comparison to packet-switching. RC-CNN can be considered as a circuit-switched network, since it provides pre-established dedicated paths (connections) for data. Here, we aim to figure out how the performance and utilization changes if a conventional packet-switched network is used to connect the PEs. To this end, we compare RC-CNN with a state-of-the-art accelerator presented in [36], which connects PEs by a packet-switched network-on-chip in order to resolve the mismatch between the accelerator and filter sizes to improve utilization. Table 9 shows the comparison results in terms of normalized

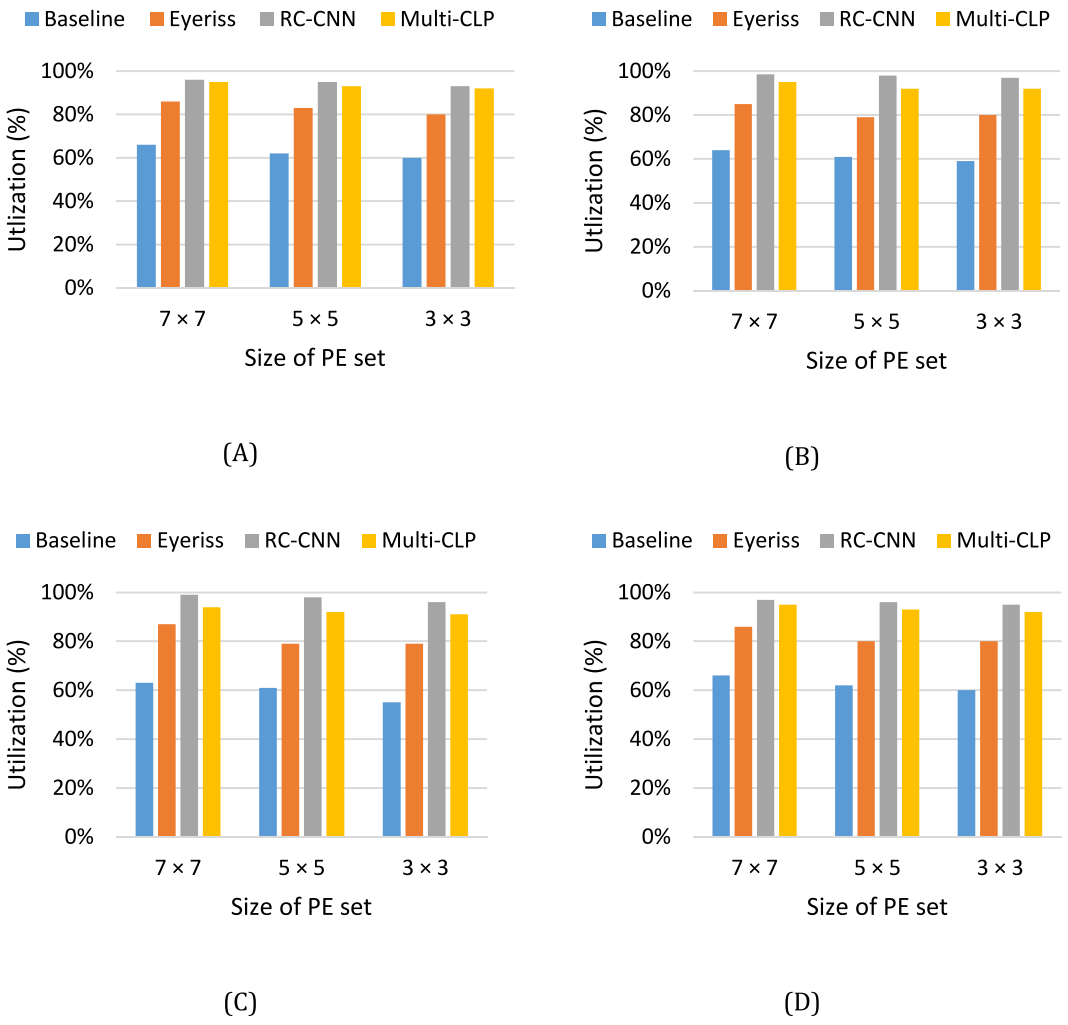

Fig. 15. Comparison of the utilization of RC-CNN and other state-of-the-art CNN accelerators under different PE set sizes (number of PEs on PE set) under (A) AlexNet (B) VGGNet (C) SqueezNet (D) GoogLeNet.

utilization, power, and latency. The accelerator in [36] uses fast 3-stage routers. In this table, the power consumption of the on-chip and off-chip memory elements in the simulator is calculated by CACTI and DRAMSim, respectively.

The packet-switched network of [36] (like any packet-switched network-on-chip) is more flexible than the circuit-switched RC-CNN. In RC-CNN, we make pre-established connections for the data and the connections remain fixed until the next reconfiguration period according to the schedule. The bandwidth of the links along the connections are dedicated to the source node of the connection, so no other PE can use the links, even if the links are idle for some cycle. Packet-switched networks, however, rely on smart routers to multiplex multiple packets on the same link, effectively handling many active connections at the same time. However, this flexibility comes at the cost of using complex routers. Reconfigurable switches, however, are made by simple switches that are by far less complex than a packet-switched router. In fact, a configuration switch removes the buffering, flow control, and arbitration operations that are essential for a packet-switched router. In our prior work, we observed that a configuration switch is up to 2x faster and up to 70% more power-efficient than a packet-switched router [41].

Fortunately, the predictable and limited number of connections in CNN dataflows, makes them less sensitive to the flexibility loss of circuit-switching. Thus, CNN traffic can benefit from the low-power and fast connections of the reconfigurable network of RC-CNN, without suffering much from its flexibility loss.

Table 9
Comparison between the packet-switching accelerator presented in [36] and RC-CNN. The numbers are normalized with respect to the packet-switching results.

Parameter name	Packet-switching [36]	RC-CNN
Utilization	1	0.91
Power Consumption	1	0.67
Latency	1	0.88

As Table 9 shows, the more flexibility of the accelerator in [36], by which all PEs can be potentially connected to each other at the same time, leads to 9% more resource utilization than RC-CNN. However, the faster switches of RC-CNN compensate for the lower utilization and even increases the latency of CNN execution by 12%. The power consumption is also decreased by more than 30%: this reduction is a result of using low-power switches in RC-CNN.

Consequently, thanks to the predictable and well-behaved traffic flow of CNNs, RC-CNN is a better choice for interconnecting PEs in a CNN accelerator than the more conventional packet-switched networks.

Sensitivity to accelerator parameters. Fig. 14 outlines the RC-CNN's PE utilization in various PE set sizes (number of PEs on PE set) for different layers of AlexNet. As the figure indicates, as the structure size grows, more utilization is achieved. The reason is the less degree of fragmentation for larger filters. Note the main capability of RC-CNN is configuring the network-on-chip in such a way that multiple filters be mapped onto the accelerator PEs. In all sizes, we keep on dimension

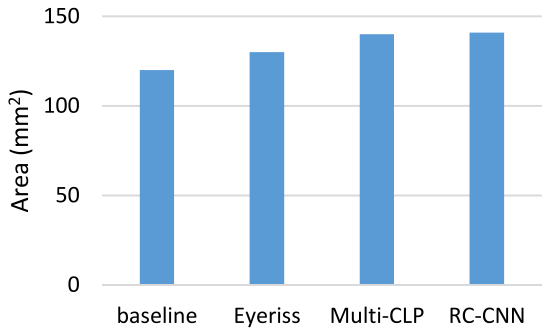

Fig. 16. Comparison of Area in the RC-CNN with other previous state-of the art architectures.

size to 11 in order to support the largest filter, which id the 11 × 11 filters of Alexnet.

Fig. 15 explores the utilization for different sizes of the PE set for the four CNN benchmarks. As Fig. 15 shows, RC-CNN outperforms the competitors across all scenarios, because it can dynamically arrange the PEs to match the filter size.

Implementation cost. To evaluate the implementation cost of RC-CNN, we implement it, as well as the other considered accelerators, in VHDL. The code is synthesized by Synopsis power compiler in a publically available 45 nm library.

Fig. 16 shows a comparison in the area between the RC-CNN with the other considered state-of-the-art architectures. Results show that a slight increase in the area due to the existence of configuration switches in the proposed architecture in comparison with previous state-of-the-art architectures. This increase in area in RC-CNN occurs due to required configuration switches in comparison with Eyeriss, Multi-CLP [35] and baseline architecture. Although the number of configuration switches in RC-CNN are higher than the number of routers in its counterpart accelerators, because of very small area of each configuration switch, the total area is not increased considerably. This increase in the area can be justified due to improvements in PE utilization and computation time parameters in the proposed architecture, which are often considered more important than area.

6. Conclusion

Convolutional Neural Networks (CNNs) have superior performance in image and pattern classification. Therefore, CNNs have a wide range of applications. However, CNNs have high computational load and memory bandwidth usage. Hardware acceleration is the primary way to overcome this ever-increasing complexity of CNNs. Most of the recent CNN accelerators arrange processing units (PEs) as a many-core accelerator architecture, with the inter-PE connections designed to the specific dataflow of the CNN layers. The performance of such accelerators is maximized when the input feature map and filter size/dimension matches that of the underlying accelerator. However, in the current fixed-size accelerator structures the same structure is used to compute CNN layers of varying dimensions which leads to sever resource underutilization. In this paper, we tackle the input feature map and filter size/dimension mismatch problem by presenting RC-CNN, a reconfigurable CNN accelerator architecture that can adapt itself to the dataflow pattern and size of the running CNN layer. RC-CNN uses a reconfigurable on-chip interconnection network that can organize a sub-set of accelerator's PEs as a PE set with the same size/dimension of the target CNN layer and customize the inter-PE connections for the layer's dataflow pattern. Since the area/energy overhead does not justify using a full-fledged packet-switched network in accelerators with fine-grained PEs, we used a reconfigurable network with very simple switches in order to efficiently implement the dynamic reconfiguration capability for many-core fine-grained CNN accelerators. Experimental results showed that RC-CNN can achieve up to 41% increase in resource utilization. It also reduced the network latency and energy consumption by 28% and 22%, respectively, compared to the state-of-the-art utilization-aware architectures that use packet-switched networks-on-chip.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Masoud Daneshtalab, Mehdi Modarressi, Hardware Architectures for Deep Learning, IET publishers, 2020.
[2] K. Chen, T. Wang, M. Ebrahimi, NoC-based DNN accelerator: A future design paradigm, in: International Symposium on Networks-on- Chip, 2019.
[3] J. Dean, D. Patterson, C. Young, A new golden age in computer architecture: Empowering the machine-learning revolution, IEEE Micro 38 (2) (2018) 21–29.
[4] V. Sze, Y. Chen, T. Yang, J. Emer, Efficient processing of deep neural networks: A tutorial and survey, Proc. IEEE 105 (12) (2017) 2295–2329.
[5] X. Hu, Y. Zeng, Z. Li, X. Zheng, S. Cai, X. Xiong, A resources-efficient configurable accelerator for deep convolutional neural networks, IEEE Access 7 (2019) 72113–72124.
[6] S.-F. Hsiao, H.-J. Chang, Sparsity-aware deep learning accelerator design supporting CNN and LSTM operations, in: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1–4.
[7] S. Das, A. Roy, K.K. Chandrasekharan, A. Deshwal, S. Lee, A systolic dataflow based accelerator for CNNs, in: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1–5.
[8] S. Xiao, W. Liu, J. Lin, Z. Yu, A data-driven asynchronous neural network accelerator, IEEE Trans. Comput. Des. Integr. Circuits Syst. (2020).
[9] V.P.K. Miriyala, K.R. Vishwanath, X. Fong, SIMBA: A Skyrmionic in-memory binary neural network accelerator, 2020, arXiv Prepr. arXiv:2003.05132.
[10] J. Sim, S. Lee, L.-S. Kim, An energy-efficient deep convolutional neural network inference processor with enhanced output stationary dataflow in 65-nm CMOS, IEEE Trans. Very Large Scale Integr. Syst. 28 (1) (2019) 87–100.
[11] B. Asgari, R. Hadidi, T. Krishna, H. Kim, S. Yalamanchili, ALRESCHA: A lightweight reconfigurable sparse-computation accelerator, in: 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020, pp. 249–260.
[12] J. Li, et al., SqueezeFlow: A Sparse CNN accelerator exploiting concise convolution rules, IEEE Trans. Comput. 68 (11) (2019) 1663–1677.
[13] F. Nasiri, H. Sarbazi-Azad, A. khademzadeh, Reconfigurable Multicast Routing for Networks on Chip, Elsevier, 2016.
[14] D. Yang, A. Ghasemazar, X. Ren, M. Golub, G. Lemieux, M. Lis, Procrustes: a dataflow and accelerator for sparse deep neural network training, in: 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020, pp. 711–724.
[15] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, X. Li, Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks, in: 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017, pp. 553–564.
[16] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, B. Yu, Recent advances in convolutional neural network acceleration, Neurocomputing 323 (2019) 37–51.
[17] S.R. Faraji, P. Abillama, G. Singh, K. Bazargan, HBUCNNA: Hybrid binary-unary convolutional neural network accelerator, in 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1–5.
[18] E. Baek, D. Kwon, J. Kim, A multi-neural network acceleration architecture, in: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), 2020, pp. 940–953.
[19] S. Ghodrati, et al., Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks, in 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020, pp. 681–697.
[20] H. Mo, et al., TFE: Energy-efficient transferred filter-based engine to compress and accelerate convolutional neural networks, in 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020, pp. 751–765.
[21] S.-C. Kao, G. Jeong, T. Krishna, ConfuciuX: Autonomous hardware resource assignment for DNN accelerators using reinforcement learning, in: 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020, pp. 622–636.
[22] B. Asgari, R. Hadidi, H. Kim, S. Yalamanchili, Eridanus: Efficiently running inference of dnns using systolic arrays, IEEE Micro 39 (5) (2019) 46–54.
[23] Arash Firuzan, Mehdi Modarressi, Masoud Daneshtalab, Reconfigurable network-on-chip for 3d neural network accelerators, 2018 Twelfth IEEE/ACM International Symposium on Networks-on-Chip (NOCS) (2018) 1–8.

[24] Arash Firuzan, Mehdi Modarressi, Masoud Daneshtalab, Reconfigurable communication fabric for efficient implementation of neural networks, 2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC). IEEE (2015) 1–8.

[25] H. Mahdiani, A. Khadem, A. Ghanbari, Mehdi Modarressi, F. Fattahi-Bayat, M. Daneshtalab, Δ NN: POwer-efficient neural network acceleration using differential weights, IEEE Micro 40 (1) (2019) 67–74.

[26] M.F. Reza, P. Ampadu, Energy-efficient and high-performance NoC architecture and mapping solution for deep neural networks, in: Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip, 2019, pp. 1–8.

[27] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel S Emer, Efficient Processing of Deep Neural Networks: A Tutorial and Survey, ieeexplore.ieee.org, 2017.

[28] S. Gudaparthi, S. Narayanan, R. Balasubramonian, E. Giacomin, H. Kambalasubramanyam, P.-E. Gaillardon, Wire-aware architecture and dataflow for CNN accelerators, in: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019, pp. 1–13.

[29] R. Guirado, H. Kwon, E. Alarcón, S. Abadal, T. Krishna, Understanding the impact of on-chip communication on DNN accelerator performance, in: 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2019, pp. 85–88.

[30] Bosheng Liu, Xiaoming Chen, Ying Wang, Yinhe Han, Jiajun Li, Haobo Xu, Xiaowei Li, Addressing the issue of processing element under-utilization in general-purpose systolic deep learning accelerators, in: Proceedings of the 24th Asia and South Pacific Design Automation Conference, 2019, pp. 733–738.

[31] Y.-H. Chen, T. Krishna, J. Emer, V. Sze, Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks, in: 2016 IEEE International Solid-State Circuits Conference (ISSCC), 2016, vol. 59, pp. 262–263.

[32] S. Sharify, et al., Laconic deep learning inference acceleration, in 2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA), 2019, pp. 304–317..

[33] E. Qin, et al., Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training, in: 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020, pp. 58–70.

[34] H. Kwon, A. Samajdar, T. Krishna, MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects, Proc. Twenty-Third Int. Conf. Archit. Support Program. Lang. Oper. Syst., 2018, pp. 461–475.

[35] Y. Shen, M. Ferdman, P. Milder, Maximizing CNN accelerator efficiency through resource partitioning, in: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), 2017, pp. 535–547.

[36] B. Zhang, H. Gu, K. Wang, Y. Yang, A novel CONV acceleration strategy based on logical PE set segmentation for row stationary dataflow, IEEE Trans. Comput. http://dx.doi.org/10.1109/TC.2021.3089366.

[37] Y.-H. Chen, J. Emer, V. Sze, Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks, ACM SIGARCH Comput. Archit. News 44 (3) (2016) 367–379.

[38] Y.-H. Chen, T.-J. Yang, J.S. Emer, V. Sze, Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices, IEEE J. Emerg. Sel. Top. Circuits Syst. 9 (2) (2019) 292–308.

[39] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. 25 (2012) 1097–1105.

[40] V. Sze, Y.-H. Chen, T.-J. Yang, J.S. Emer, How to evaluate deep neural network processors: TOPS/W (alone) considered harmful, IEEE Solid-State Circuits Mag. 12 (3) (2020) 28–41.

[41] Mehdi Modarressi, Arash Tavakkol, Hamid Sarbazi-Azad, Application-aware topology reconfiguration for on-chip networks, IEEE Trans. Very Large Scale Integr. (VLSI) Systems 19 (11) (2011) 2010–2022, http://dx.doi.org/10.1109/TVLSI.2010.2066586.

[42] M. Gao, X. Yang, J. Pu, M. Horowitz, C. Kozyrakis, Tangram: Optimized coarse-grained dataflow for scalable nn accelerators, in: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 807–820.

[43] Nasrin Akbari, Mehdi Modarressi, A High-Performance Network-on-Chip Topology for Neuromorphic Architectures, IEEE International Conference on Embedded and Ubiquitous Computing (EUC) (2017).

[44] J.R. Stevens, A. Ranjan, D. Das, B. Kaul, A. Raghunathan, Manna: An accelerator for memory-augmented neural networks, in: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019, pp. 794–806.

[45] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, T. Lillicrap, One-shot learning with memory-augmented neural networks, 2016, arXiv Prepr. arXiv:1605.06065.

[46] H. Kwon, A. Samajdar, T. Krishna, Rethinking NoCs for spatial neural network accelerators, in: 2017 11th IEEE/ACM Int. Symp. Networks-on-Chip, NOCS 2017, 2017, p. 19.

[47] R. Zhao, et al., Accelerating binarized convolutional neural networks with software-programmable fpgas, in: Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2017, pp. 15–24.

[48] S. Liang, S. Yin, L. Liu, W. Luk, S. Wei, FP-BNN: BInarized neural network on FPGA, Neurocomputing 275 (2018) 1072–1086.

[49] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, X. Ji, High-performance fpga-based cnn accelerator with block-floating-point arithmetic, IEEE Trans. Very Large Scale Integr. Syst. 27 (8) (2019) 1874–1885.

[50] J. Guo, et al., AccUDNN: A GPU memory efficient accelerator for training ultra-deep neural networks, in: 2019 IEEE 37th International Conference on Computer Design (ICCD), 2019, pp. 65–72.

[51] Y. You, A. Buluç, J. Demmel, Scaling deep learning on gpu and knights landing clusters, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2017, 1–12.

[52] A. Jafari, M. Hosseini, A. Kulkarni, C. Patel, T. Mohsenin, BiNMAC: Binarized neural network manycore accelerator, in: Proceedings of the 2018 on Great Lakes Symposium on VLSI, 2018, pp. 443–446.

[53] K. Qiu, et al., ResiRCA: A resilient energy harvesting ReRAM crossbar-based accelerator for intelligent embedded processors, in: 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020, pp. 315–327.

[54] K. Zou, Y. Wang, H. Li, X. Li, Learn-to-scale: Parallelizing deep learning inference on chip multiprocessor architecture, in: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2019, pp. 1172–1177.

[55] Vivienne SZE, Designing hardware for machine learning: The important role played by circuit designers, IEEE Solid-State Circuits Mag. 9 (4) (2017) 46–54.

[56] A. Yasoubi, Reza Hojabr, Takshi H., Mehdi Modarressi, M. Daneshtalab, CuPAN–High Throughput on-Chip Interconnection for Neural Networks, Springer.

[57] N.P. Jouppi, In-datacenter performance analysis of a Tensor Processing Unit, in: Proc. 44th Annu. Int. Symp. Comp. Archit., 2017, pp. 1–12.

[58] H. Kwon, A. Samajdar, T. Krishna, A communication-centric approach for designing flexible DNN accelerators, IEEE Micro 38 (6) (2018) 25–35.

[59] T. Krishna, A communication-centric approach for designing flexible DNN accelerators, in: Proceedings of the 12th International Workshop on Network on Chip Architectures, 2019, p. 1.

[60] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, Lingli Wang, A high performance FPGA-based accelerator for large-scale convolutional neural networks, in: 2016 26th International Conference on Field Programmable Logic and Applications (FPL), 2016, pp. 1–9.

[61] Fengbin Tu, Weiwei Wu, Yang Wang, Hongjiang Chen, Feng Xiong, Man Shi, Ning Li, Jinyi Deng, Tianbao Chen, Leibo Liu, Shaojun Wei, Yuan Xie, Shouyi Yin, Evolver: A deep learning processor with on-device quantization-voltage-frequency tuning, IEEE J. Solid-State Circuits (2020).

[62] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, Saibal Mukhopadhyay, Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory, Ieeexplore.Ieee.Org, 2016.

[63] K. Simonyan, Very deep convolutional networks for large-scale image recognition, 2014, arxiv.org. arXiv:1409.1556.

[64] C. Szegedy, et al., Going Deeper with Convolutions, cv-foundation.org, 2014, pp. 1–9.

[65] https://neurohive.io/wp-content/uploads/2018/10/AlexNet-1.png.

[66] F.N. Iandola, S. Han, M.W. Moskewicz, K. Ashraf, W.J. Dally, K. Keutzer, SqueezeNet: AlexNet-Level accuracy with 50x fewer parameters and <0.5 MB model size, 2016, arXiv Prepr. arXiv:1602.07360.