Chain-NN: An Energy-Efficient 1D Chain Architecture for Accelerating Deep Convolutional Neural Networks

Shihao Wang, Dajiang Zhou, Xushen Han, Takeshi Yoshimura
Graduate School of Information, Production and Systems, Waseda University, Kitakyushu, Japan
wshh1216@moegi.waseda.jp

Abstract—Deep convolutional neural networks (CNN) have shown their good performances in many computer vision tasks. However, the high computational complexity of CNN involves a huge amount of data movements between the computational processor core and memory hierarchy which occupies the major of the power consumption. This paper presents Chain-NN, a novel energy-efficient 1D chain architecture for accelerating deep CNNs. Chain-NN consists of the dedicated dual-channel process engines (PE). In Chain-NN, convolutions are done by the 1D systolic primitives composed of a group of adjacent PEs. These systolic primitives, together with the proposed column-wise scan input pattern, can fully reuse input operand to reduce the memory bandwidth requirement for energy saving. Moreover, the 1D chain architecture allows the systolic primitives to be easily reconfigured according to specific CNN parameters with fewer design complexity. The synthesis and layout of Chain-NN is under TSMC 28nm process. It costs 3751k logic gates and 352KB on-chip memory. The results show a 576-PE Chain-NN can be scaled up to 700MHz. This achieves a peak throughput of 806.4GOPS with 567.5mW and is able to accelerate the five convolutional layers in AlexNet at a frame rate of 326.2fps. 1421.0GOPS/W power efficiency is at least 2.5x to 4.1x times better than the state-of-the-art works.

Keywords—convolutional neural networks; CNN; accelerator; ASIC; power efficiency; memory bandwidth

I. INTRODUCTION

Convolutional neural networks (CNN) have recently been a hot topic as they have led to great breakthrough in many computer vision tasks. Many CNN-based algorithms like AlexNet [1] and VGG [2] have proved to be stronger than conventional algorithms. Among these networks, convolutions usually account for the major part of the total computations in both training and testing phases. Further, there is a trend that CNN is towards deeper and larger for a better performance [4]. For example, ResNet [3] has recently proved a network with more than a thousand layers can further enhance the performance compared to a shallower network.

The increasing network scalability results in more computational complexities. It usually costs quite a long time to train deep CNNs and also challenges the design of real-time CNN applications. Therefore, CNN-specific accelerators are desired. The accelerators are expected to have not only a high reconfigurability to always achieve a high performance for various CNN parameters, but also the good energy and area efficiency for deployments on battery-limited devices.

To achieve these goals, researchers have been exploring efficient CNN accelerators on various platforms. The designs on CPU or GPU like [5] can achieve a high reconfigurability at the expense of high energy costs and usually suffer from the Von-Neumann bottleneck which limits their performance. Architectures on FPGA [6-7] or ASIC [8-15] are good candidates as these architectures can be totally customized to specific high-performance and energy-efficient needs. However, it usually involves high design complexities for supporting a high reconfigurability which will reversely affect the performance or efficiency. For example, [12] builds a 2D PE array, which has an on-chip network and inter-PE communications. Thus, an efficient architecture is desired to balance these metrics including the energy efficiency, for the increasing popularity of CNNs.

In this paper, we introduce our energy-efficient 1D chain architecture called Chain-NN for CNN accelerators, which contains the following contributions:

- We give a taxonomy of existing CNN accelerators to figure out their pros and cons.
- A novel energy-efficient 1D chain architecture is given. Its dedicated dual-channel architecture and column-wise scan input pattern provides energy-friendly reusability of input data to achieve 1.4TOPS/W power efficiency.
- 1D chain architecture of Chain-NN also supports a high reconfigurability by 84-100% of PE utilization rate for mainstream CNNs with low design complexities.
- Experimental results in TSMC 28nm show maximum 806.4GOPS throughput at 700MHz. Compared to state-of-the-art works, it achieves 2.5x to 4.1x power efficiency.

The rest of the paper is organized as follows. Sec. II briefly introduces CNN algorithms. In Sec. III, we give a taxonomy of existing CNN accelerators and introduce the features of our Chain-NN. Sec. III, we give a detailed discussion on the proposed 1D chain architecture. The experimental results are shown in Sec. IV. Finally, we conclude this paper in Sec. V.

II. CNN BACKGROUND

Convolutional neural networks are trainable architectures where convolutional layers are the primary computational parts inside a network. Each layer receives input feature maps (iifmaps) and generates the output feature maps (oofmaps), which represent some particular features as shown in Fig. 1.
Fig. 1. A generalized example of computations in CNN convolutional layers.

We may regard ifmaps/ofmaps as 3D structures composed of multiple 2D channels. For each ofmaps channels, convolution is done between the 3D ifmaps and correspondent 3D kernels. Moreover, several group of 3D ifmaps compose of a batch, where the same group of kernels are shared. Therefore, convolutional layers can be regarded as 4D computations as shown in the Equation (1).

\[
ofmaps[n][m][x][y] = \text{bias}[m] + \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} \text{ifmaps}[n][c][x+i][y+j] \times \text{Kernel}[m][c][i][j]\]

The meanings of indexes are shown in Table I. These parameters vary for different layers and different CNNs.

### III. HIGH LEVEL DESCRIPTION OF CHAIN-NN

#### A. Taxonomy of Existing CNN Architectures

We classify existing accelerator architectures into two categories according to how data are used inside processor cores. The way of data usage affects not only the reconfigurability, but also the required memory bandwidths (the amount of data movements). According to [16], the data movements of convolutional operands can be more expensive than ALU operations under the existing techniques. We give a detailed description and conclusion of the pros and cons of these two categories, followed by introducing how Chain-NN can improve over the existing works in Sec. III.B.

1) Memory-centric Architectures

As shown in Fig. 2(a), these architectures rely on the addressability of memories to achieve its reconfigurability for various CNN parameters. The processor core is simply a stacking of PEs. There is no data storage or data reuse paths inside the processor. A sophisticated central controller is necessary to decide when processor core should communicate with memories and which part of memories are accessed during the processing period.

These architectures can be reconfigured to adapt for various CNNs while the efficiency is sometimes unsatisfactory. They highly rely on memories for huge amount of data movements like [9,10,13]. Some achieve their reconfigurability by utilizing fewer PEs with fewer feeding data from memories, resulting in a waste of computing resources [8,15].

2) 2D Spatial Architectures

2D spatial architectures in Fig. 2(b) can reduce the data movements from memories by reusing them inside the processor core. Each PE can not only do basic computations, but also maintain a local controller to communicate with other PEs. PE usually contains local scratch pad memories to store data that are frequently used during a certain period. 2D spatial architectures divide the central controller in memory-centric ones and distribute them into each PE.

This solution can reduce the amount of data movements and have been employed in many works like [11,12]. However, their results show that the power and area cost of peripheral circuits for inter-PE communications can’t be ignored. Moreover, this architecture has to consider the constraints in two dimensionalities for deployments, which sometimes limits its reconfigurability and scalability.
B. High-level Description of Chain-NN

This paper introduces Chain-NN, a novel energy-efficient 1D chain architecture. As shown in Fig. 2(c), the PEs are organized as a chain architecture. Chain-NN is controlled by a finite state machine which changes its states according to a specific dataflow. The execution procedure is like this: 1) The finite-state machine is initialized to specific CNN parameters. 2) It starts to load related kernels into the processor core. 3) The ifmaps are continuously streamed into Chain-NN and convolution results are calculated. This paper mainly focuses on the design of processor core of Chain-NN and we leave the design exploration of memory hierarchy as the future work.

We conclude that Chain-NN has following advantages. First, it has a good energy efficiency compared to the others. It has a chain of dedicated dual-channel PEs which transfer data to adjacent PEs for data reuses with low control complexities. This not only reduces the total amount of memory accesses, but also shortens the distance of data fetching from on-chip SRAM level to PE level. Secondly, it has a high reconfigurability to be capable of achieving a high performance for various CNN parameters. This is due to its one dimensional organization of PEs, whose constraints are greatly relaxed compared to the 2D spatial architectures. An instantiation of our Chain-NN has proved to achieve an 84-100% PE utilization ratio considering the mainstreaming CNN parameters. Finally, this architecture involves fewer overheads when scaled up to a higher parallelism or clock frequency. This feature not only brings efficiencies of both power and area, but also makes the architecture flexible in matching various demands of designers.

IV. CHAIN-NN: 1D CHAIN ARCHITECTURE

A. 1D Chain Architecture

1D chain architecture is the processor core of Chain-NN. It consists of numbers of cascading PEs. A group of adjacent \( K^2 \) PEs forms a systolic primitive for convolution computations. An example of their relationship is shown in Fig. 3. We will give a detailed description on the 1D primitives in Sec. IV.B and the dual-channel PE architecture in Sec. IV.C.

In Fig. 3, the PE chain is cut into 1D primitives according to the kernel size. The upper part shows the case where each 1D primitive contains 9 PEs \((K=3)\) while the lower refers \( K=2 \) case. The first and last PE in a primitive involve the ports to communicate with memory hierarchy, thus a set of primitive ports are attached.

The peak throughput of accelerator is proportional to the number of active PEs/primitives in the architecture. As a case study, we assume a systolic chain contains 576 PEs. The

<table>
<thead>
<tr>
<th>Kernel Size</th>
<th># of PEs of primitive</th>
<th># of active primitives</th>
<th># of active PEs</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>3x3</td>
<td>9</td>
<td>64</td>
<td>576</td>
<td>100%</td>
</tr>
<tr>
<td>5x5</td>
<td>25</td>
<td>23</td>
<td>575</td>
<td>99.8%</td>
</tr>
<tr>
<td>7x7</td>
<td>49</td>
<td>11</td>
<td>539</td>
<td>93.6%</td>
</tr>
<tr>
<td>9x9</td>
<td>81</td>
<td>7</td>
<td>567</td>
<td>100%</td>
</tr>
<tr>
<td>11x11</td>
<td>121</td>
<td>4</td>
<td>484</td>
<td>84.0%</td>
</tr>
</tbody>
</table>

B. 1D Systolic Primitive for 2D Convolution

To support 1D chain architecture, each primitive for convolutions should also be designed as 1D implementation. This is done by pipelining a chain of multiply-accumulate operations (MAC) in Fig. 4 to form a systolic architecture like [16], which we call 1D systolic primitive. Notice that other pipelining schemes may produce more efficient architectures and we will discuss them in the future. In this paper, we mainly focus on how systolic primitives are employed inside the proposed 1D chain architecture for eliminating data locality.

Before that, we first present some design details about the 1D systolic primitives. It consists of a group of \( K^2 \) identical PEs which is shown in Fig. 4(a). The \( K^2 \) PEs are mapped to a convolutional kernel window. Thus, each PE is in charge of a 16-bit fixed-point MAC operation with a specific kernel weight.

Instead of reading the data in parallel, data are streamed in and go through every PE along the ifmaps path in Fig. 4(b). This guarantees an invariant input bandwidth requirement regardless of the kernel size \( K \). Whenever a new pixel is streamed in at time \( t \), previous \((K-1)\) pixels during \([t-K^2+1, t-1]\) and this pixel constitute a 2D convolution window in ifmaps.
Fig. 5. The input order of ifmaps affects the PE throughput. Dual-channel convolutional windows. It means that at least three cycles are required for fetching the rest three pixels due to the single channel (one ifmaps pixel per cycle). The 1D primitive has to be idle during this period. Therefore, this case shows that one-channel PE architecture can only achieve 1/K (33% in this case) of the peak throughput.

Therefore, we introduce the dual-channel PE architecture for this matching issue. Fig. 5(b) shows how we can benefit from it. We first give a limitation that it requires K adjacent rows of ifmaps to be processed simultaneously. Then, (2K-1) rows of ifmaps (we refer this as input pattern in the following) are streamed in through two channels at the timestamps shown inside each pixel. Two channels are in charge of odd and even columns respectively. By doing this, we can find pixels of \([t-K^2+1, t]\) form a convolution window for any given \(t\). Moreover, the pixel orders inside any window follow the column-wise scan order. We call this column-wise scan input pattern and it supports dual-channel PE can continuously start new convolutional operations after the initialization stage. Thus, we can 100% utilize the computational resources with the overheads of adding only one new ifmaps channel.

The dual-channel PE architecture is shown in Fig. 6. The dual channels (OddIF and EvenIF) inside each PE are in charge of transferring ifmaps into next PE. A multiplexer decided which channel’s data is sent to MAC. Specifically, OddIF is in charge of odd columns (shown as blue numbers) while EvenIF only cares about the even columns (shown as orange numbers). EvenIF starts working after \((K+1)\) cycle delay than OddIF. Meanwhile, a set of primitive input ports and function units (gray blocks in Fig. 6) are employed to support the systolic chain in Sec. IV.A. Inside each PE, there is a RegisterFile-based internal storage (kMemory) for storing the stationary kernels. The control of kMemory is designed to follow the dataflow. Moreover, we emphasize that we can easily pipeline the MAC path to shorten the critical path for a higher clock frequency, as shown by the red line in Fig. 6.

Dataflow and Memory Hierarchy

In a recent work [7], a detailed analysis on CNN memory-efficient dataflow and on-chip memory hierarchy have been discussed. We utilize their strategies for testing our proposed 1D chain architecture. The dataflow is modified to support the column-wise scan input pattern requirement as discussed in

C. Dual-channel PE Architecture

This part presents the dual-channel PE architecture and the column-wise scan input pattern. The dual-channel means there are two data feed channels along the ifmaps path in Fig. 4(b). It can guarantee the high performance with the invariant input bandwidth of systolic primitives.

We first explain why a single channel based PE architecture can’t fully utilize the computational resources of the 1D primitive. Let’s still take \(K=3\) as an example in Fig. 5(a). We have mentioned the matching issue that only if the order of \([t-K^2+1, t]\) ifmaps pixels accurately matches the stationary kernel window can the systolic primitive start to perform a convolution operation. However, inside the 2D ifmaps shown in Fig. 5(a), at most six pixels are overlapped for any two convolutional windows. It means that at least three cycles are
Sec. IV.C. The detailed dataflow is shown by a loop structure in Fig. 7. The parallelism of Chain-NN unrolls the loop of ofmaps and the convolutions, marked as ParaTile. Meanwhile, two separate memories, iMemory and oMemory, form the memory hierarchy for data reusing among InnerTile.

V. EXPERIMENTAL RESULTS

A. Methodology

Chain-NN has been coded by SystemVerilog HDL and synthesized with TSMC 28nm HPC library under slow operation conditions (0.81V, 125°C) using the Synopsys Design Compiler. Layout is done by Encounter as shown in Fig. 8. We implemented a float-point-to-fix-point simulator which is integrated with MatConvnet for generating the test dataset including convolutional layers of pre-trained networks for MNIST, Cifar-10, AlexNet and VGG-16. Verification is done using ModelSim by both function simulation and post-synthesis simulation. The output results of hardware are checked with the simulator results on-the-fly. The power of the Chain-NN is analyzed by Power Compiler under typical operation conditions (0.9V, 25°C). We generate switching activity interchange format (SAIF) by simulating the post-synthesis designs for power simulation.

B. Performance

We instantiate a Chain-NN with 576 PEs, each of which is pipelined into three stages so that the critical path delay is reduced to 1.428ns (700MHz). Theoretically, this design can achieve a peak throughput of 806.4GOPS.

AlexNet is used to evaluate the realistic performance. We use totally 352KB on-chip memories, including kMemory in each PE, to support the data reuse in AlexNet based on the dataflow in Fig. 7. In detail, we implement 32KB for iMemory, 295KB for kMemory and 25KB for oMemory. In terms of kMemory, 295KB are averagely distributed into 576 PEs, resulting in a capable of 256 kernel weights per PE.

C. Energy Efficiency

Energy efficiency is measured by the throughput per watt. Fig. 10 shows Chain-NN consumes 567.5mW and contributes 806.4GOPS, achieving a power efficiency of 1421.0GOPS/W. In details, around 90% of the power consumption is from the 1D chain architecture including kMemory while only 10.55% is cost by the memory hierarchy. The power consumption of memories is greatly reduced in Chain-NN from two aspects. Firstly, we have shortened the length of most data movements from on-chip SRAM level to PE level. In detail, dedicated PEs with column-wise scan input pattern guarantee ifmaps are reused $K^2$ times averagely inside systolic primitives. This can reduce the data movements of ifmaps.
Comparison with the State-of-the-art Works

Table V shows the comparison results. Our Chain-NN is at least 2.5x energy efficient compared to [10]. We roughly divided the design into two parts: processor core and memory hierarchy. If only processor cores are considered, [10] can achieve around 3.0TOPS/W while Chain-NN is around 1.7TOPS/W. The reduced paths in Chain-NN are utilized more efficiently. Secondly, the fewer memory bandwidth requirements can partially simplify the design complexity of memory hierarchy. These contribute to the 1.7 times area efficiency.

**ACKNOWLEDGMENT**

This work is supported by Waseda University Graduate Program for Embodiment Informatics (FY2013-FY2019).

**REFERENCES**


![Fig. 10. The comparison of power efficiency with DaDianNao [10].](image)