Stream Processing Dual-Track CGRA for Object Inference

Xitian Fan, Di Wu, Wei Cao, *Member, IEEE*, Wayne Luk, *Fellow, IEEE*, and Lingli Wang, *Member, IEEE*

***Abstract*— With the development of machine learning technology, the exploration of energy-efficient and flexible architectures for object inference algorithms is of growing interest in recent years. However, not many publications concentrate on a coarse-grained reconfigurable architecture (CGRA) for object inference algorithms. This paper provides a stream processing, dual-track programming CGRA-based approach to address the inherent computing characteristics of algorithms in object inference. Based on the proposed approach, an architecture called stream dual-track CGRA (SDT-CGRA) is presented as an implementation prototype. To evaluate the performance, the SDT-CGRA is realized in Verilog HDL and implemented in Semiconductor Manufacturing International Corporation 55-nm process, with the footprint of 5.19 mm^2 at 450 MHz. Seven object inference algorithms, including convolutional neural network (CNN), *k*-means, principal component analysis (PCA), spatial pyramid matching (SPM), linear support vector machine (SVM), Softmax, and Joint Bayesian, are selected as benchmarks. The experimental results show that the SDT-CGRA can gain on average 343.8 times and 17.7 times higher energy efficiency for Softmax, PCA, and CNN, 621.0 times and 1261.8 times higher energy efficiency for *k*-means, SPM, linear-SVM, and Joint-Bayesian algorithms when compared with the Intel Xeon E5-2637 CPU and the Nvidia TitanX graphics processing unit. When compared with the state-of-the-art solutions of AlexNet on field-programmable gate array and CGRA, the proposed SDT-CGRA can achieve a 1.78 times increase in energy efficiency and a 13 times speedup, respectively.**

***Index Terms*— Acceleration, coarse-grained reconfigurable architecture (CGRA), deep learning, domain-specific computing, object inference.**

I. Introduction

WITH the breakthrough of deep learning technology in speech applications [2], computer vision [3], and other tasks in artificial intelligence [4], the architecture exploration of related algorithms is a hot research topic in terms of energy efficiency and flexibility. For example, Google's tensor processing unit (TPU) is built specifically for machine learning acceleration and tailored for the TensorFlow software framework [5], [6]. It is reported that TPU can achieve an order of magnitude energy efficiency enhancement compared with the traditional approaches [5], [6]. In academia, deep learning accelerator, such as DaDianNao [7], ShiDianNao [8], Eyeriss [9], Cambricon [10], and so on, has shown impressive improvements in energy efficiency compared with general-purpose processors and graphics processing units (GPUs). However, application-specific accelerators (e.g., DaDianNao, ShiDianNao, and Eyeriss) are often tailored to specific algorithms [e.g., convolutional neural network (CNN)]. The hard-wired logic prohibits them to migrate from one algorithm to another. As a result, they may not be proper approaches to accelerate algorithms in an object inference flow, since a general object inference flow not only contains CNN but also includes other algorithms, as shown in Fig. 1. As for the application-specific instruction processors (ASIPs), e.g., Cambricon, their energy efficiency is restricted due to the logic and memory overhead to support flexibility. On the contrary, coarse-grained reconfigurable architecture (CGRA), a promising paradigm for domain-specific computing, has been shown that it can outperform ASIPs in energy efficiency and has more flexibility than application-specific accelerators [11]. However, few CGRA architectures concentrate on the domain of object inference in machine learning.

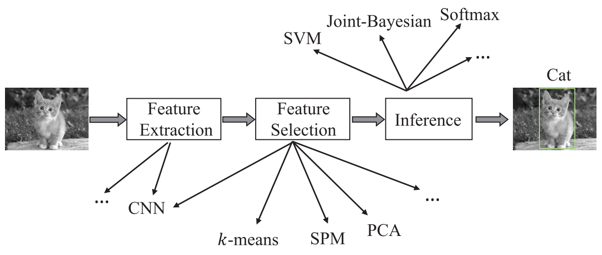


Fig. 1. General flow of object inference.

The increasing demand to process data streams more efficiently for data-centric applications for an embedded system is another motivation to design an appropriate CGRA. As shown in Section III, the computing processes in an object inference flow have three inherent characteristics. The first is stream processing, which means that an algorithm can be divided into multiple computing kernels while each kernel computes in the stream manner. The second is fixed kernel-level operations (OPs), which means that the computing patterns of the kernels remain unchanged in a long execution period (even up to thousands of execution cycles), such as the image filtering and the gradient computation of image in the edge detection. The one is large amount of memory storage requirements for input data, intermediate data, and output results.

Manuscript received April 12, 2017; revised September 17, 2017 and December 6, 2017; accepted January 16, 2018. Date of publication February 12, 2018; date of current version May 22, 2018. This work was supported by the National Natural Science Foundation of China under Grant 61131001. This paper is an extended version of earlier work presented at the 26th International Conference on Field-Programmable Logic and Applications (FPL2016). *(Corresponding authors: Wei Cao; Lingli Wang.)*

X. Fan, D. Wu, W. Cao, and L. Wang are with the State Key Laboratory of Application Specific Integrated Circuit and System, Fudan University, Shanghai 201203, China (e-mail: xtfan14@fudan.edu.cn; 15110720011@fudan.edu.cn; caow@fudan.edu.cn; llwang@fudan.edu.cn).

W. Luk is with the Department of Computing, Imperial College London, London SW7 2AZ, U.K. (e-mail: wl@doc.ic.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2018.2797600

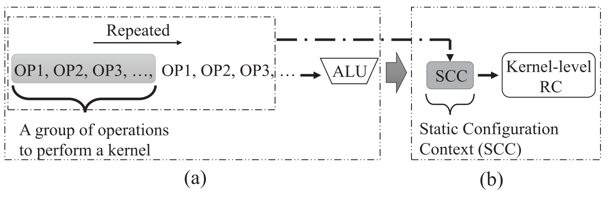

Repeated
OP1, OP2, OP3, ...,
OP1, OP2, OP3, ...
ALU
A group of operations to perform a kernel
SCC
Kernel-level RC
Static Configuration Context (SCC)
(a)
(b)

Fig. 2. (a) Loading OPs in every execution cycle. (b) Static configuration.

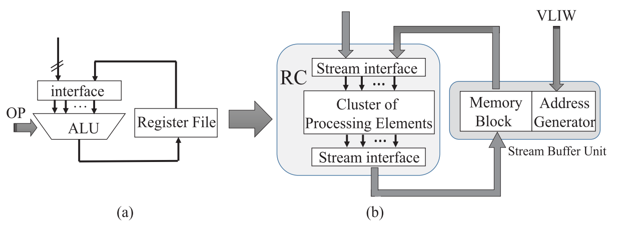


Fig. 3. Migration from (a) word-level granularity to (b) stream-level granularity.

To design a CGRA that caters to these characteristics, we first use a cluster of processing elements [e.g., arithmetic logical unit (ALU) and multiplier] as an elementary reconfigurable cell (RC) for kernel-level OPs. This design choice is made, because a complex cell has more possibilities to map a complex kernel in a single cell rather than across multiple cells. As a result, the cost in data transmission among multiple cells for complex kernel OPs can be reduced, and hence, the computation can be speeded up. This approach is similar to expression-grained reconfigurable array (EGRA) [12] and FPCA [13], which have proved that the clustered RC array can outperform traditional CGRAs that can only map one computing OP in each cell. Second, stream processing is adopted as programming paradigm. Due to the characteristic of the fixed kernel-level OPs, we propose a dual-track programming model based on the stream processing. The key idea is to adopt static configuration to construct the functionalities of RCs for kernel-level OPs and to apply dynamic configuration to manage data streams transmission. This approach is similar to the differentiation between the configuration and the I/O data transfers in Morphsys [14]. However, the major difference is that our model applies static configuration to fix the RCs' functionalities and local interconnections in a long execution period, while Morphsys loads the OPs from context memory for their RCs in every execution cycle, as shown in Fig. 2(a). According to the characteristic of the fixed kernel-level OPs in object inference applications, the group of OPs for a specific kernel is repeatedly executed. In this case, the repeated OPs can be simplified as static configuration in our approach [as shown in Fig. 2(b)], which can reduce the bandwidth to load OPs and decrease the overall power consumption. Third, with the granularity of input data increasing from word level to stream level, the intermediate storage (e.g., register file in traditional reconfigurable block) of each RC should also be increased from word-level granularity to stream-level granularity. As a result, stream memory is provided for each RC. This approach can also meet with the requirements of increasing intermediate storage in object inference flow. Fig. 3 shows this migration from word-level granularity (register file) to stream-level granularity [stream buffer unit (SBU)].

To summarize, our contributions in this paper are as follows.

1) A novel CGRA-based approach to accelerate algorithms in a general object inference flow is provided. This approach has three aspects. First, it adopts stream processing, such that each kernel-level RC computes in stream manner. Second, it uses both static configuration and dynamic configuration for stream processing, such that static configuration constructs kernel functionalities, while dynamic configuration is used for scheduling of data streams. Third, stream memory is used for buffering the input, output, and intermediate streams.
2) A CGRA implementation prototype, called stream dual-track CGRA (SDT-CGRA), is provided to realize the above-mentioned approach. Novelties include the composable and decomposable RC architectures that are interconnected in a reverse-S topology and stream-driven control mechanism to simplify control behavior for the cluster-based RC architecture, as shown in Section IV-D.

The rest of this paper is organized as follows. Section II covers the previous work in this domain. Section III presents the design approach based on the analysis of algorithms on the object inference flow. Section IV introduces an architecture called SDT-CGRA to implement the proposed design approach. Section V includes several examples to demonstrate the mapping strategies. Section VI contains the experimental results. Finally, Section VII discusses future work and concludes this paper.

II. Related Work

A. Architecture Perspective

In the past decades, several CGRA frameworks are proposed. Among them, ADRES [15] is one of the widely studied templates based on data flow computing. The tightly coupled characteristic with a host processor allows multiple customized instructions to be efficiently executed on ADRES. As a result, instruction-level parallelism can be exploited to accelerate algorithms. Another CGRA template is Morphsys [14], which is organized by 2-D mesh homogeneous reconfigurable cells in single instruction multiple data fashion. It exploits data-level parallelism to accelerate applications.

These two approaches exploit parallelism from different perspectives to improve the computing performance for the applications in a specific domain. However, for the algorithms in an object inference flow, the characteristic of fixed kernel-level OP makes ADRES and Morphsys no longer be energy-efficient. For example, both ADRES and Morphsys have to load context instructions in every execution cycle, which is not necessary according to the characteristic of the fixed kernel-level OPs in our applications.

On the contrary, DySER [16], a CGRA architecture that explores functionality and parallelism specialized in a single array has shown that specializing the common data paths in their proposed architecture with certain execution periods for the programs can improve the energy efficiency. We further extend this specialized approach to the object inference domain, where the kernel-level OPs are suitable for specialization.

Kernel-level OPs in stream processing require kernel-based processing units in order to compute efficiently. EGRA [12], an expression-level granularity CGRA framework, has showed that the expression-level or kernel-level CGRA fabric can outperform traditional CGRA approaches. In this paper, reconfigurable block in kernel-level granularity is used as the elementary reconfigurable cell (RC). The major difference is that we derive the RC architecture and its computing manner according to the computing characteristics of our target applications.

BilRC [17] and elastic CGRA [18] can be categorized into another CGRA template. Both of them are similar to commercial (field-programmable gate array) FPGA in architecture arrangement and configuration manner. For BilRC, applications' dataflows are mapped statically and scheduled dynamically by execution triggering, while elastic CGRA depends on elastic interconnection [19] to manage dataflows. In our approach, elastic interconnection is adopted for data transmission among RCs and global memory, interconnections in RC units.

An earlier version of this paper was presented in [1]. This paper further optimizes the architecture of RC and SDT-CGRA. More specifically, the number of ALUs, first-input, first-ouputs (FIFOs), and static configuration bits are reduced by 20%, 20%, and 38.5%, respectively. Details will be discussed in Section IV. More complementary experiments are provided in Section VI.

B. Application Perspective

Most of the CGRA architectures proposed in the past mainly concentrate on the digital signal processing. For example, ADRES, BilRC, and elastic CGRA mainly target at the acceleration of fast Fourier transform, discrete cosine transformation/inverse discrete cosine transformation, and so on. In [20], a CGRA architecture is proposed to accelerate video decoding in multiple standards. In [13], an architecture called FPCA is designed for medical image processing.

As for machine learning, MAPLE [21] introduces an FPGA-based reconfigurable accelerator for classification. They abstract all selected algorithms as matrix multiplications and design a matrix multiplication engine for all of them. Actually, this approach limits the exploitation of locality properties for some machine learning algorithms, e.g., CNN. In [22], a CNN acceleration approach based on CGRA (called EMAX) is proposed. However, limited experiments are done to evaluate the performance of EMAX on CNN. In [23], a multithread CGRA (called M-CGRA) is proposed to accelerate CNN only. However, the object inference flow not only contains CNN but also includes other traditional algorithms. If one wants to design an architecture for CNN acceleration only, the algorithm-specific approaches, such as Eyeriss [9], deep neural network processing unit [24], deep neural architecture [25], and hybrid-neural-network processor [26], are more suitable and energy efficient.

III. Computing Characteristics Analysis

A. Algorithms Characteristics

Object inference is one of the most important topics in computer vision. In the competitions of PASCAL VOC [27] and ImageNet [28], a general processing flow of the proposed algorithms in the past several years can be abstracted into three key stages: feature extraction, feature selection, and inference, as shown in Fig. 1. In this paper, several representative algorithms covering these three stages are selected as case studies. Among them, CNN is used for feature extraction; k-means, spatial pyramid matching (SPM) [29], and principal component analysis (PCA) are adopted for feature selection; the linear support vector machine (SVM), Softmax [3] and Joint Bayesian [30] are used for inference. Table I summarizes these algorithms, while their computing patterns are analyzed and shown in Table II.

TABLE I
Selected Representative Algorithms

Stage	Representative Algorithms
Feature Extraction	Convolutional Neural Network (CNN)
Feature Selection	k-means, SPM, PCA
Inference	SVM, Softmax, Joint-Bayesian

TABLE II
Summary of Major Computation Patterns From Representative Algorithms in General Object Inference Flow

Pattern Description	Pattern Equation	Related Algorithm
conv	$\sum_i a_i \times b_i$	CNN
interpolation	$a_i \times c + b_i, c_i <= c < c_{i+1}$, $i=1,2,...,N$	CNN, Softmax
div	$\frac{a}{b}$	CNN
sqrt	$\sqrt{a}$	k-means
distance	$\sum_i \lvert a_i - b_i \rvert$ $\sum_i (a_i - b_i)^2$	k-means
matrix multiplication	$A_{a_1 \times a_2} B_{a_2 \times b_2}$	CNN, SVM, PCA, Joint-Bayesian, Softmax
histogram	$\sum_i (I_i == K_j ? w : 0)$	SPM

Generally, the detection of objects requires to process images or feature maps by scanning OPs within specific processing scopes in multiple scales. Each scanning OP can be regarded as a kernel function executing over a limited scope of the input image/feature map and then shifting to the next scope in a specific order, as shown in Fig. 4(a). As for multiscale detection, general methods include classifiers running on the pyramid of images/feature maps (e.g., deformable part model [31]), the pyramid of filters running on the feature maps (e.g., a speedup robust feature [32]), and the pyramid of referenced bounding boxes on the final regression functions (e.g., faster regions with CNN feature [33]). For simplicity, multiscale detection can be regarded as multilayer scanning OP, as shown in Fig. 4(b).

Each process in Fig. 4 has the following inherent properties. The functionality of the computing kernel remains unchanged for the same input image/feature map. The only difference is the input data for the computing kernel. As a result, any specific kernel will execute repeatedly over all the correspondent kernel scopes. In this process, the computing patterns of the kernels remain unchanged for a long execution periods, which correspond to the fixed kernel-level OPs. In this case, the input

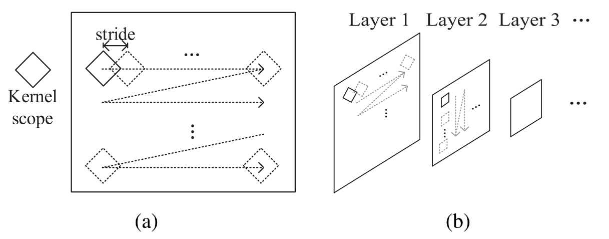


Fig. 4. Computation procedure of a kernel over a scope. (a) Scanning behavior example. (b) Illustration of multiscale scanning.

data from all the kernel scopes can be organized as bunches of data streams, while the execution of a kernel is in stream processing manner. Besides, the multiscale/multilayer OPs have pyramid input images/feature maps, or pyramid output of intermediate results, which means that the storage requirement (includes intermediate storage) is much critical than CGRAs for other applications, e.g., digital signal processing. In this case, another computing characteristic of algorithms in object inference is the requirement of sufficient storage. To summarize, the computing characteristics of algorithms in the object inference flow are as follows:

1) stream processing;
2) fixed kernel-level OPs;
3) ample storage.

In our selected representative algorithms, kernel-level OPs in stream processing account for most of computations. In CNN, take AlexNet [3] for example, 92% of the computation are convolution and pooling. The rest, which includes the fully connected layer (contains matrix multiplication pattern in Table II) and the softmax layer (contains matrix multiplication and interpolation pattern in Table II), can also be organized as kernel-level OPs in stream processing manner. More importantly, the computing pattern of a specific layer, e.g., the first convolutional layer, can remain unchanged for nearly one million execution cycles. In this case, if we consider the power consumption in loading each OP for an ALU in every execution cycle in Fig. 2(a), the power saving of our static configuration approach is substantial.

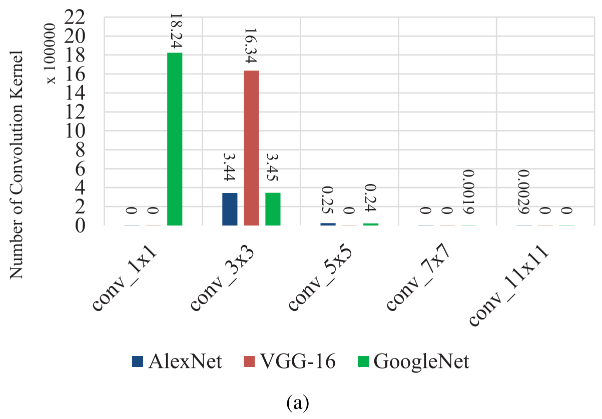
B. Guidelines for Architecture Design

As discussed in Section III-A, the computing characteristics can enable us to design a CGRA in the following approaches.

1) Design kernel-level granularity RCs in stream manner.
2) Increase the size of intermediate storage.
3) Use static configuration for RCs to construct their functionalities and dynamic configuration for data streams scheduling.

The first two steps explain how to design a CGRA architecture for our target applications, while the last one, which is called the dual-track programming model in this paper, explains how the architecture works. We will discuss these two parts in detail in the following.

1) How to Design a CGRA Architecture for Target Applications: Convolution and matrix multiplication are two of the most important computing patterns in algorithms of object inference flow. These two computing patterns involve many multiplication–accumulation OPs. As a result, the elementary RC should be based on the multiplier-ALU



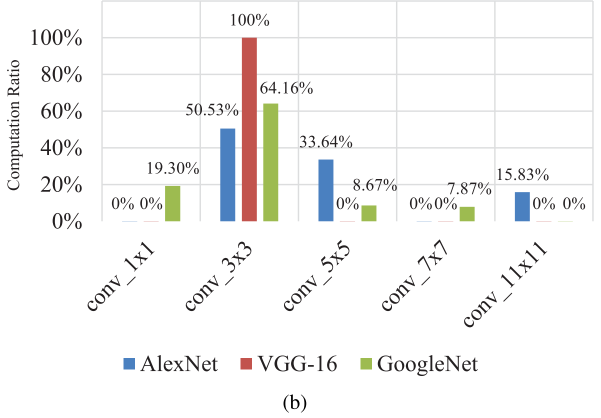


Fig. 5. Information of convolution kernels in AlexNet, VGG-16, and GoogleNet. (a) Number of convolution kernels with different sizes in three CNN architectures. (b) Computing workload ratios of different sizes of convolution kernels in all convolutional layers for specific CNN architecture.

(MUL-ALU for short) units. An approach, such as placing multipliers and ALUs in different parts, e.g., BilRC [17], would increase the routing resource requirements and even lead to the placement or routing conflict. As a result, the tightly coupled multiplier and ALU for our applications is a proper approach.

On the other hand, extra OPs are required to support further additions, accumulations, or logic OPs in convolution, matrix multiplication, distance calculation, and so on. If single MUL-ALU unit is used as the elementary RC, the extra additions, accumulations, or logic OPs will lead to low utilization ratio of multipliers in MUL-ALU units. As a result, combining MUL-ALU units and extra ALUs as the elementary RC is necessary.

We studied the problem of determining the number of MUL-ALU units and extra ALUs in each RC by collecting run time statistics over several representative CNN algorithms, since CNN is the most important part in terms of computation and inference accuracy. Fig. 5(a) shows the statistics of different sizes of convolution kernels in AlexNet, VGG-16 [34], and GoogleNet [35]. It is clear that the number of the convolution of 3×3 (denoted as *conv*_3×3) ranks number one in AlexNet and VGG-16, while GoogleNet has the most number of the 1×1 convolution kernels (denoted as *conv*_1×1). However, from the perspective of computation, the 3×3 convolution kernel still ranks number one in three CNN architectures

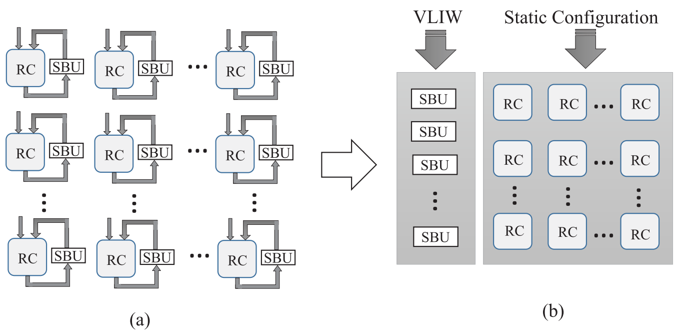


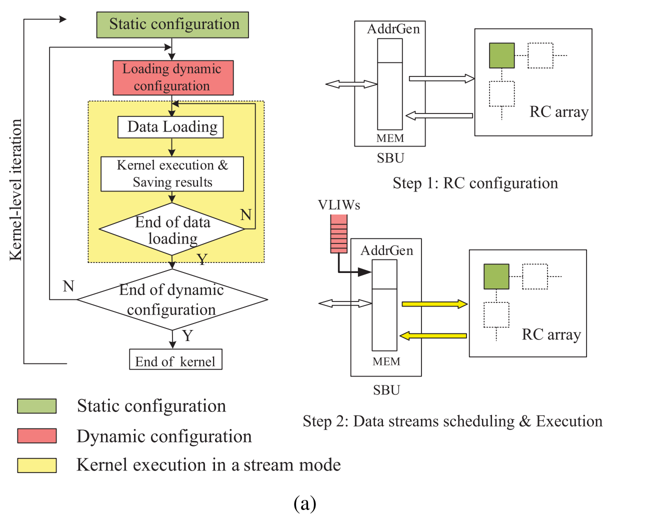
Fig. 6. (a) RC array. (b) RC array reorganization.

[see Fig. 5(b)]. In VGG-16, the computation ratio of *conv*_3×3 in all the convolutional layers is even up to 100%. As a result, efficient support for 3 × 3 convolution kernel can be regarded as the foundation to design an RC unit. In this case, three MUL-ALU units and an extra ALU are combined together as the elementary RC in our CGRA approach. As for other sizes of convolution kernel and other computation patterns, flexible interconnections inside and outside RCs are provided to support them. Details will be discussed in Section IV.

The whole CGRA architecture can be regarded as a 2-D array of RC units, as shown in Fig. 6(a). In order to introduce the static and dynamic configurations clearly, we can reorganize the architecture in Fig. 6(a) as Fig. 6(b), where all the SBUs are lined up as one column. In our dual-track programming model, the SBUs are controlled by the dynamic context in a very long instruction word (VLIW) format, while the RC array is configured statically according to the computing kernels. The CGRA architecture based on the stream processing, dual-track programming model is also called SDT-CGRA in this paper.

2) How the Architecture Works: The RC array in Fig. 6(b) is guided by the dual-track programming model, whose configuration flow is shown in Fig. 7(a). For each computing kernel in a kernel-level iteration, the functionality of given computing kernel is initially constructed by static configuration. Then, the data stream manager is configured according to the scheduling requirements, such as loading data from the off-chip memory to the SBU or from the SBU to the RC array and storing data from the RC array to the SBU or from the SBU to the off-chip memory. After the configurations, the address generator in the SBU starts to generate addresses to issue load or store OPs [indicated by the innermost loop in Fig. 7(a)], while the RC array performs as a consumer as well as a producer in this process. It is worth mentioning that the dynamic VLIW context only supports load and store OPs.

To illustrate the proposed model, a convolution example, which is widely used in image processing and machine learning domain, is presented. Fig. 7(b) shows the process of static mapping and dynamic scheduling of the data streams for the convolution OP. For simplicity, we assume that the input image *Imap* contains two rows of data, denoted as *L1* and *L2*. The size of the convolution kernel is 1 × 3. Based on the dual-track model, the convolution OP is statically mapped into one RC, e.g., RC1. The corresponding scheduling of the data streams is supposed to be compiled into two VLIWs: Instruction 1 and Instruction 2. Each of them can issue two



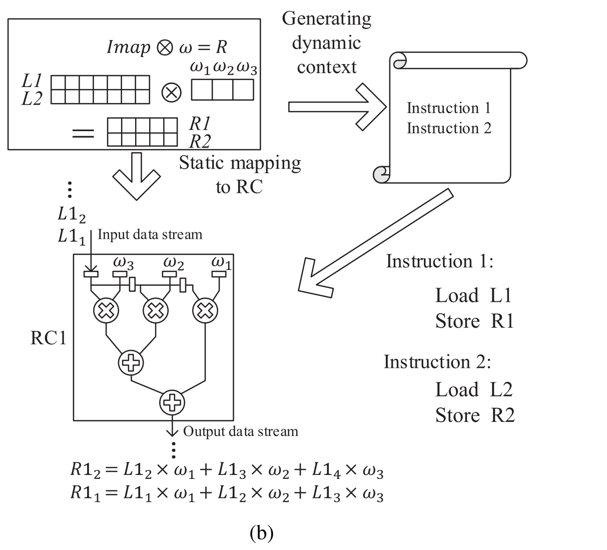

$Imap \otimes \omega = R$
$\omega_1\omega_2\omega_3$
L1
L2
R1
R2
Generating dynamic context
Instruction 1
Instruction 2
Static mapping to RC
$L1_2$
$L1_1$
Input data stream
ω_3 ω_2 ω_1
RC1
Output data stream
Instruction 1:
Load L1
Store R1
Instruction 2:
Load L2
Store R2
$R1_2 = L1_2 \times \omega_1 + L1_3 \times \omega_2 + L1_4 \times \omega_3$
$R1_1 = L1_1 \times \omega_1 + L1_2 \times \omega_2 + L1_3 \times \omega_3$
(b)

Fig. 7. Programming model of the SDT-CGRA and an example of convolution OP based on the proposed dual-track model. (a) Programming model. (b) Example for the model.

concurrent OPs, which are used to load the input data and store back the results. For example, in Instruction 1, *L1* is read out from the SBU and then sent to RC1. At the same time, the output stream result *R1* is stored back to the SBU as soon as it is available. Instruction 2 performs the same OPs except that the input data stream and the output stream are different. In this method, RC1 is configured to be a stream processing unit for efficient convolution computation over the input image. To demonstrate the benefits of the dual-track programming model for SDT-CGRA, we assume that two configuration methods in Fig. 2 are applied in RC1, respectively. It can be supposed that both methods need to load the same length of configuration contexts (denoted by LCC) to calculate one convolution result, e.g., $R1_1$ in Fig. 7(b). As for the approach that requires to load OPs in every execution cycle, the total length of configuration contexts loaded to ALU is 10 × LCC (the final result R in Fig. 7(b) contains ten elements). As for our static configuration approach, the total length of configuration contexts is still to be LCC, since we only need to configure the RC1 in one time. From this point of

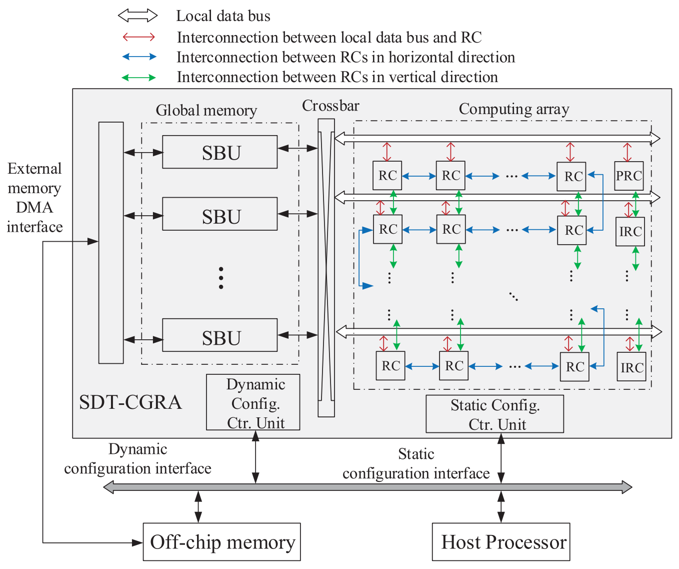

Local data bus
Interconnection between local data bus and RC
Interconnection between RCs in horizontal direction
Interconnection between RCs in vertical direction
Global memory
Crossbar
Computing array
External memory DMA interface
SBU
RC
PRC
IRC
SDT-CGRA
Dynamic Config. Ctr. Unit
Static Config. Ctr. Unit
Dynamic configuration interface
Static configuration interface
Off-chip memory
Host Processor

Fig. 8. Typical acceleration system consists of an SDT-CGRA architecture, an off-chip memory, and a host processor.

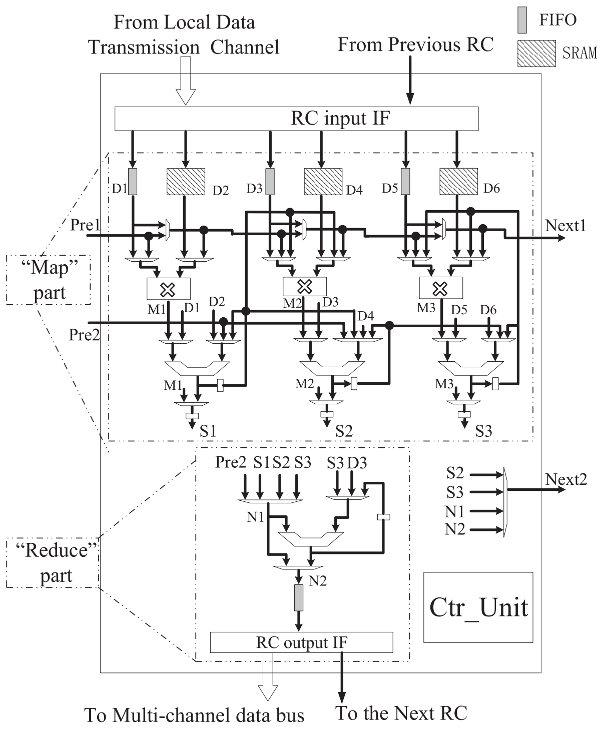


Fig. 9. Microarchitecture of RC from the data path perspective.

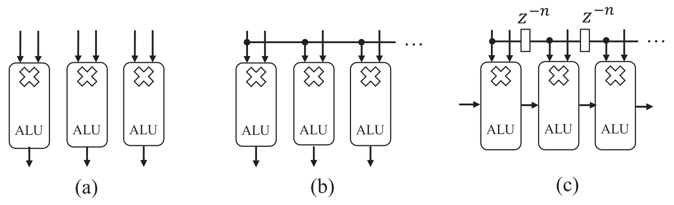


Fig. 10. Three different work styles of MUL-ALU units. z^n means to delay n clock cycles of the data. (a) Independent manner. (b) Broadcast manner. (c) Systolic manner.

view, the static configuration is better. If we consider the power consumption of loading configuration contexts, the advantages of our approach are more evident.

IV. SDT-CGRA Architecture: A Prototype

According to the design strategies introduced in Section III, we present a prototype to implement SDT-CGRA. Each part of SDT-CGRA is introduced as follows.

A. Overview of SDT-CGRA

The top-level architecture of the proposed SDT-CGRA and a typical system is shown in Fig. 8. The SDT-CGRA unit can be mainly organized into a global memory section and a computing array section according to the difference of configuration manner. The global memory section is used to cache data streams and issue load or store OPs through dynamic configuration. In contrast, the computing array section, which works in static configuration manner, consists of several columns of RC and one column of special RC [shown as interpolation RC (IRC) and power RC (PRC) in Fig. 8]. Special RCs are used to support some special OPs, such as power function (corresponding to PRC) and transcendental function that can be approximated by interpolation (corresponding to IRC). Since such OPs account for small computation generally, only several special RC units are provided. Details of these units are introduced in Section IV-C.

In addition to the memory blocks and the computing units, the interconnections among them play a key role in data transmission and selection of mapping strategies. For example, the interconnections marked as blue arrows in Fig. 8 are used to connect the RC units in reverse-S topology, which is designed to provide the composition and decomposition capability among adjacent RCs in the horizontal direction. As a result, the larger computing kernel that exceeds the computing volume of one RC can be realized by several RCs and helps to reduce the idle processing elements. Details can be found in Section IV-D. The green arrows, on the other hand, can enable the results from one RC to pass directly to the next RC in the vertical direction. This type of direct data transmission method is inspired by the fast data relay CGRA [36] to reduce the global memory communication congestion. The local data bus is designed for data transmission between SBUs and RCs. With the help of a scalable crossbar switch [37], each SBU can be accessed by any RC, IRC, and PRC based on the local data transmission channels.

There are three interfaces in total for the proposed SDT-CGRA. The external direct memory access interface provides a direct access to the off-chip memory for the SBUs. The remaining two interfaces are used for configuration: one for static configuration and the other for loading dynamic context instructions.

B. RC Architecture

1) Detailed Architecture: The guidelines in Section III-B show that the RC architecture can be designed with three MUL-ALU units and an extra ALU for our target applications. Fig. 9 shows a detailed RC architecture in SDT-CGRA. Three MUL-ALU units can be configured to execute multiply–accumulate OPs, distance calculations, or other computing patterns in independent manner, broadcast manner, or systolic manner, as shown in Fig. 10. An extra ALU is used to perform further addition, accumulation, or other logic OPs to reduce the bandwidth of output interface. This idea follows [38], where it is called the "*map-reduce*" structure. In our architecture, three MUL-ALU units belong to the "*map*" part, while the remaining ALU belongs to the "*reduce*" part (see Fig. 9). The "*map*" part can be used to execute concurrent OPs, while the "*reduce*" part is used to collect the results from the "*map*" part. The "*map*" part is composable and decomposable, which will be introduced in Section IV-D. When compared with the RC architecture in our previous work [1], the numbers of FIFOs and ALUs are both reduced from 5 to 4 (20% reduction) without any impact on the mapping of computing kernels. Besides, the number of multiplexers, which provide internal interconnections to efficiently support other computing patterns, is reduced from 37 to 22 (40.5% reduction). In addition, the static configuration bits for multiplexers are reduced by 38.5%. The intention to design such tightly coupled RC unit with three MUL-ALU units is to increase the OP intensity per input data, which can help to improve the computation performance according to the roofline model [39].

To implement stream processing, the input and output stream interfaces in Fig. 3 are realized with two types of local memories [FIFOs and static random access memories (SRAMs)]. FIFOs are used to maintain the working status of Ctr_Unit by the "full" and "empty" signals. The SRAMs are adopted to cache the data that are used frequently, e.g., the weights of convolution kernels. In many cases, the SRAM can also be used to perform as a double buffer to overlap the time cost in data transmission from SBU to RC by the time consumed in computing.

2) Stream-Driven Control Mechanism: The control mechanism of RC is determined by two characteristics involving stream processing and static configuration. To accommodate stream processing, the FIFOs in input and output stream interfaces and the interconnection channels based on elastic interconnection [19] (see Section IV-D) are provided to issue "processing flag." That is to say, if the input FIFOs are not empty and the output FIFO is not full, or the input interconnection channels have valid data and the output interconnection channels are writable, the Ctr_Unit in Fig. 9 will control the RC to process the input data. Otherwise, the Ctr_Unit will stop the RC from processing. This stream-driven control mechanism of Ctr_Unit is realized by finite-state machines and counters, which is configured in static configuration stage and designed to generate control signals, such as read enable signals for input FIFOs, read addresses for input SRAMs, write enable signal for the output FIFO, and clear signals for ALUs when they are configured to be accumulators, ready and valid signals in interconnection channels for adjacent RCs, and so on.

C. Special RC Design

Two types of special RCs, PRC and IRC, are developed to support the power functions and piecewise functions, respectively. Consider the first PRC, designed to calculate $x^{1/2}$, $x^{-1/2}$, and x^{-1} based on the fast inverse square root

Algorithm 1 Calculate $y = x^p$ [40] in C Language Syntax

```
Input: $x, p, -1 \leq p \leq 1, x > 0$, $x$ is in floating-point format
Output: $y = x^p$
1: float $xorig = x$;
2: int $i = *(\text{int}*)x$;
3: float $j = (1.0 - p) * 1064975338 + (p * i)$; // 1064975338
   is called magic number
4: $i = \text{int}(j)$;
5: $x = *(\text{float}*)i$;
6: $y = (1.0 - p) * x + p * pow(x, (p - 1)/p) * xorig$;
```

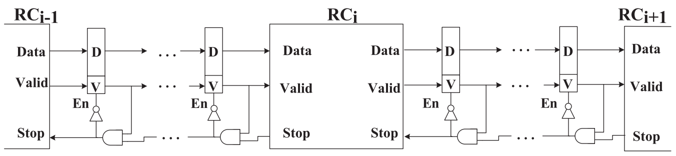


Fig. 11. Elastic control mechanism [19].

algorithm [40] shown in Algorithm 1. Many multiplications in Algorithm 1 can be simplified to be shifting or addition OPs, while $pow(x, (p-1)/p)$ can be calculated by multiplications when p sets to be $1/2$, $-1/2$, or -1. Due to the requirement of the floating-point representation of the input value x according to the fast inverse square root algorithm, the input x is first converted from the fixed-point format to the floating-point format. In line 6 of the code in Algorithm 1, all the values required to calculate y will be converted back to the fixed-point value to calculate the final result in order to reduce the hardware complexity.

Consider next IRC, designed to calculate the interpolation for transcendental functions that can be approximated by piecewise functions, as shown by the following expression:

$$f(x) = a_i \times x + b_i, \quad x \in [x_i, x_{i+1}), \quad i = 0, 1, \ldots, N-1. \tag{1}$$

Specifically, the input data x are compared with the boundary values of each interval from x_0 to x_{N-1} in parallel to generate an address for lookup tables. Then, the coefficients of a_i and b_i from lookup tables are used to calculate the interpolation result $f(x)$. It is worth mentioning that both PRC and IRC are independent of RCs. As a result, the data transmissions between them are through SBUs and local data buses.

D. Interconnections

The interconnections of SDT-CGRA are organized into two types. The first is the interconnections between RCs, while the second is the crossbar between SBUs and the RC array. The main strategy for these interconnections is the elastic data transmission mechanism (see Fig. 11), which can be used to simplify the control procedure by converting dynamic scheduling to dataflow control [19]. The "*stop*" and "*valid*" signals determine the handshaking process and maintain the reliable data transmission between different nodes.

1) Interconnections Among RCs: A simple example is used to introduce the functionality of the reverse-S interconnections among RCs. Assuming that five MUL-ALUs are required to implement an expression E, three MUL-ALUs in the first

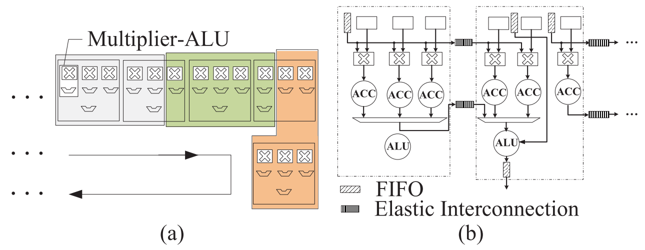


Fig. 12. (a) Illustration of the RC decomposition and combination in different rows. (b) Illustration of elastic interconnection to support RC decomposition and combination, some ALUs are configured to be accumulators (ACC).

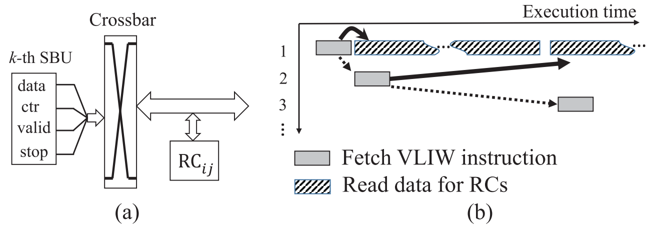
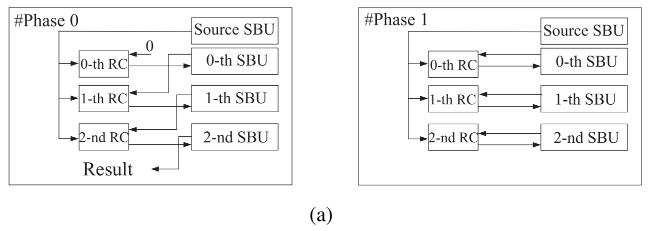


Fig. 13. (a) Crossbar switch. (b) Illustration of the dynamic configuration and execution relationship.



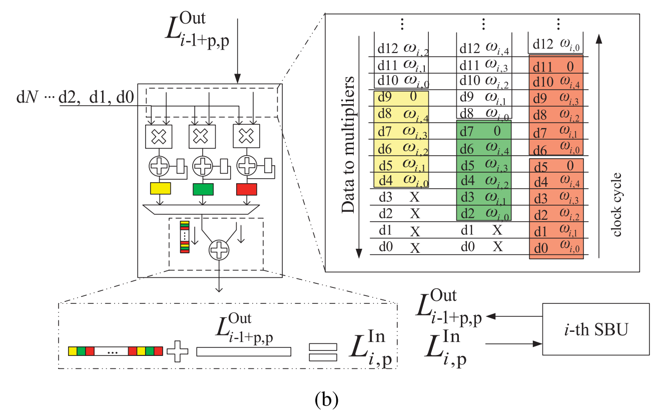


Fig. 14. (a) Example of mapping 5 × 5 convolution kernel into three RCs, which includes two working phase. Each phase is correspondent to a dynamic configuration instruction. (b) The computation process of phase $p(p = 0, 1)$.

RC and two MUL-ALUs in the second RC can be combined together to map E. The unused resources of the second RC can be further combined with other RCs. Fig. 12(a) shows such an approach, where five RCs in two adjacent rows are configured to calculate three such expressions in parallel.

Fig. 12(b) shows the details of the decomposition and combination process. The elastic interconnections are used to transfer the input data and the results that are required by the next RC. Although the second RC is split into two parts, its control behavior is still independent of other RCs as a result of elastic interconnections. That is to say, the working status of the second RC just depends on the "*valid*" and "*stop*" signals of the elastic interconnections as well as the "*full*" and "*empty*" signals of the FIFOs in the input and output interfaces. No other control signals from other RCs are required.

2) Scalable Crossbar Switch: The scalable crossbar switch in SDT-CGRA performs as a bridge to interconnect the SBUs and the RC array. It provides the capability for all the RCs to access each SBU. To accommodate the characteristics of stream processing and dynamic data stream scheduling, the crossbar switch is controlled dynamically by each select signal along with data stream in each input channel, as indicated by "*ctr*" in Fig. 13(a).

E. SBU Architecture

As indicated in Fig. 3, each SBU contains a memory block and an address generator that can provide read and write addresses simultaneously. The OPs of the address generator in each SBU, such as read/write from/to the RC array or the off-chip memory, are controlled by dynamic configuration contexts. To demonstrate the control flow of the data streams, suppose that the kth SBU issues several write OPs to the jth RC in the ith row [denoted as RC_{ij} in Fig. 13(a)]. In the first VLIW cycle, the kth SBU and the control signals corresponding to the output channel in the crossbar are configured. After the end of configuration, the address generator in the kth SBU starts to generate addresses to read data from the memory block for RC_{ij}. At the same time, the next VLIW instruction is fetched and waits for the finish of the current instruction, as shown in Fig. 13(b). This double-buffer technique adopted in the dynamic configuration process can help to reduce the configuration overhead.

V. Algorithm Mapping Examples

In machine learning, convolution OP and large-scale matrix multiplication are two of the most common computing patterns. For example, the convolutional layers and the fully connected layers account for nearly 92% and 8% computation workloads, respectively, in AlexNet. In this section, several strategies are demonstrated to map computing kernels to the proposed SDT-CGRA based on the convolution OP and the matrix multiplication. It should be mentioned that these strategies are only part of the computing methods in the considered algorithms; other methods can also be mapped on SDT-CGRA.

A. Mapping Strategies of the Convolution Layers

One of the strategies to map convolution OPs in the convolutional layers can support various sizes of convolution kernels with arbitrary strides. Without loss of generality, suppose that the mapped kernel is 5 × 5, the stride is 2 × 2, and the width of the 2-D input feature map is N. The number of MUL-ALUs that are required to map such a convolution kernel is determined by $\lceil 5/2 \rceil \times \lceil 5/2 \rceil$. There are two working phases (depend on the stride of the kernel) for data scheduling, where each phase corresponds to one dynamic VLIW configuration instruction. Fig. 14 shows the mapping and computing strategies. The convolution kernel is mapped into three RCs with two working phases, as shown in Fig. 14(a). Before the computation begins, the weights of the convolution kernel are loaded into three SRAM blocks in each RC. To simplify the control procedure, the size of the convolution kernel is extended to be 6×6 by padding 0s at the right and bottom sides of the kernel. In this case, the weights in the first two rows of the extended convolution kernel, $(w_{0,0}, w_{0,1}, \ldots, w_{0,4}, 0)$ and $(w_{1,0}, w_{1,1}, \ldots, w_{1,4}, 0)$, are loaded into all the SRAM blocks in the first RC. The weights of the third and the fourth rows are loaded into the second RC, while the last two rows are loaded into the last RC. After the weights are initialized, the input data are read from the source SBU in the row order and broadcasted to all the mapped RCs.

As for the even rows (row number starts from 0) in the input feature map, three RCs work in #Phase 0. For the odd rows, the mapped RCs work in #Phase 1. The computation process of both phases is shown in Fig. 14(b), where $L^{\text{In}}_{i,p}$ and $L^{\text{Out}}_{i-1+p,p}$ represent the input and output of intermediate results buffered in the ith SBU at phase p, $i = 0, 1$, and 2; $p = 0$ and 1; and $i - 1 + p = -1$ *means* $L^{\text{Out}}_{i-1+p,p} = 0$. For example, the first row of the input data is loaded into the zeroth RC to convolve with the first row of the weights. The corresponding results, denoted as $L^{\text{In}}_{0,0}$, are buffered in the zeroth SBU. After the end of #Phase 0, the zeroth RC switches to work in #Phase 1 to compute the convolution of the second row of the input data and the second row of the weights. At the same time, the results from #Phase 0 (denoted as $L^{\text{In}}_{0,0}$ earlier) are read from the zeroth SBU (denoted as $L^{\text{Out}}_{0,1}$) and sent to the zeroth RC to add up with the current convolution results to generate $L^{\text{In}}_{0,1}$. This process is repeated until all the data of the input feature map are sent to the RCs. We can see that the input data are fully reused and the convolutions are computed in parallel.

Generally speaking, suppose that the size of convolution kernel is $k_x \times k_y$ with the stride of $s_x \times s_y$, then $\lceil k_x/s_x \rceil \times \lceil k_y/s_y \rceil$ MUL-ALUs are needed according to this mapping strategy. The decomposition and combination characteristics of RCs introduced in Section IV-D provide the capability to map larger convolution kernels flexibly and efficiently.

B. Mapping Strategy for the Fully Connected Layers

Matrix multiplication is the major computing pattern in each fully connective layer. For example, the OPs in the first fully connected layer of AlexNet can be expressed as

$$O_{1\times4096} = F_{1\times9216} \times \omega_{9216\times4096}. \tag{2}$$

If the input feature F from the previous layer is directly used to compute the output result O, the system suffers from frequent loading of the weight ω from the SBU to SRAM blocks in RCs. In this case, the overheads of transmission of the weights will lead to the decline of overall performances. One idea is to reuse the weights as much as possible. A direct method is to adopt a "batch" strategy, which means that a certain number of input features, e.g., 100, are batched together to construct a larger matrix, such as $F_{100\times9216}$. In this way, the weights can be reused 100 times so that the time cost to load new weights into the SRAM blocks

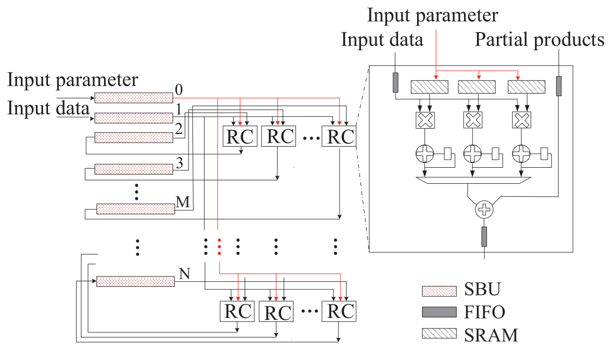


Fig. 15. Mapping strategy of large-scale matrix multiplication.

can be hidden by the computing time. Due to the capacity limitation of each SRAM block, the weight matrix has to be divided into several smaller submatrices so that each one can be loaded into one SRAM block. In order to adopt the double-buffer strategy, the dimension of each submatrix should not exceed half of the size of an SRAM block, e.g., 128. As a result, the first dimension of the weight matrix can be divided into 72 parts ($72 \times 128 = 9216$). Similarly, the input feature matrix $F_{100\times9216}$ is divided into 100×72 blocks. Each block is denoted as $F^{i,j}_{1\times128}$, where i and j are the block indices. Suppose there are 25 RCs in SDT-CGRA. The number of SRAM blocks in all RCs is 75. The size of the second dimension of the weight matrix is not divisible by 75 ($4096 = 54 \times 75 + 46$). As a result, the weight matrix is divided unequally into two types. One is t_1: $\omega^{m,n}_{128\times75}$ and the other is $t2 : \omega^{m,n}_{128\times46}$, where m and n are the indices. After the partition, (2) can be expressed as

$$O_{100\times4096} = \begin{bmatrix} F^{0,0} & \cdots & F^{0,71} \\ F^{1,0} & \cdots & F^{1,71} \\ \vdots & \vdots & \vdots \\ F^{99,0} & \cdots & F^{99,71} \end{bmatrix} \times \begin{bmatrix} \omega^{0,0}_{t1} & \cdots & \omega^{0,53}_{t1} & \omega^{0,54}_{t2} \\ \omega^{1,0}_{t1} & \cdots & \omega^{1,53}_{t1} & \omega^{1,54}_{t2} \\ \vdots & \ddots & \vdots & \vdots \\ \omega^{71,0}_{t1} & \cdots & \omega^{71,53}_{t1} & \omega^{71,54}_{t2} \end{bmatrix}. \tag{3}$$

To illustrate the computing process of (3) on the SDT-CGRA, we take the multiplication process of $F_{100\times9216}$ with $\omega^{0,0}_{t1}$ as an example. The computing process mapped into the SDT-CGRA is shown in Fig. 15. First, 75 columns of $\omega^{0,0}_{t1}$ are loaded into 75 SRAMs in the RC array. Then, the submatrices $F^{0,0}$, $F^{1,0}$, ..., $F^{99,0}$ in the first column of $F_{100\times9216}$ are broadcast to all the RCs one by one for processing with multiplication and accumulation. The results are sent to the SBUs and then added up with the products of $\omega^{1,0}_{t1}$ and the second column of $F_{100\times9216}$. This process is repeated until the final matrix result is computed.

VI. Experimental Results

A. Evaluation Setup

In order to evaluate the proposed architecture, seven algorithms shown in Table I (see Section III) are selected as benchmarks. The typical implementations and problem sizes are listed in Table III. For example, we take k-means and SPM

TABLE III
TYPICAL IMPLEMENTATIONS, APPLICATIONS, AND PROBLEM SIZES OF THE SELECTIVE ALGORITHMS

Algorithm	Implementation	Problem Size
CNN	Caffe based on MKL and cuBLAS	AlexNet: $227 \times 227 \times 3$
k-means	MKL and cuBLAS	Vocabulary size:200 Feature dimension: 128
PCA	MKL and cuBLAS	Input dimension: 320000 Output dimension: 150
SPM	MKL and cuBLAS	Pyramid layer : 3 Vocabulary size:200
Softmax	Caffe based on MKL and cuBLAS	Class number: 1000 Feature dimension: 4096
SVM	MKL and cuBLAS	Feature dimension: 3780
Joint Bayesian	MKL and cuBLAS	Feature dimension: 150

TABLE IV
DETAILED INFORMATION OF EACH UNIT

Unit Type	Information	
	Details	Number
RC	16-bit fixed-point	5×5
PRC	4× 16-bit fixed-point multipliers 2× 16-bit fixed-point adders 1× 32-bit fixed-point adder 2× 32-bit fixed-point shifters	2
IRC	16-bit fixed-point	3
FIFO	16-bit×64, 128 Byte	4 for each RC, 3 for each PRC, 2 for each IRC
SRAM in RC	16-bit×256, 512 Byte	3
SBU	1 SRAM, 1 Address Generator	27
SRAM in SBU	16-bit ×1024, 2 KByte	27

from [29] for object detection with the vocabulary size of the codebook to be 200. The evaluations of these algorithms on different platforms are all based on the same problem size. It is worth noting that all the algorithms are evaluated for inference, while the training stage is not evaluated in this paper.

To map the algorithms into the SDT-CGRA, the static configurations and the dynamic VLIW instructions are written in the microcode format manually. In order to reduce the effort of writing static and dynamic configurations, we encapsulate the static configurations of the most common computing patterns and the SBU read/write OPs into a library. To map a given task into the SDT-CGRA, it is programmed with the provided application program interfaces of the library.

B. Implementation Details

The proposed SDT-CGRA contains 5×5 (25) RCs, five special RCs, 54-kB global SRAM (27 SBU), and 54.6-kB local memory (including FIFOs and SRAMs). The detailed information is shown in Table IV, where all the computing units are designed based on 16-bit fixed point except several 32-bit fixed-point adders and shifters in PRCs. The whole SDT-CGRA is implemented in Verilog HDL and then synthesized, placed, and routed with the Synopsys design compiler and the IC compiler based on the Semiconductor Manufacturing International Corporation (SMIC) 55-nm library.

The final results reported by the IC compiler show that the area of the proposed architecture is 5.19 mm^2. The average chip-only power consumption (dynamic power plus static power) of SDT-CGRA is evaluated based on the simulation wave files over the seven selected benchmarks. The results show that the average power dissipation is 0.84 W. The breakdown of the area, the average power dissipation, and the chip-only energy consumption are listed in Table V, where we can see that RC array accounts for 65% of the chip area and 62.3% of the average power dissipation. The SBU ranks the second place in all metrics. Besides, the delay of the critical path is 2.21 ns, which means that the architecture can run at 450 MHz. Since the SDT-CGRA contains 86 multipliers

TABLE V
CHARACTERISTICS OF THE LAYOUT OF SDT-CGRA AND THE AVERAGE POWER DISSIPATION AND TOTAL ENERGY CONSUMPTIONS OVER SEVEN BENCHMARKS

Component	Area (μm^2)	Chip-only Power (W)	Chip-only Energy (mJ)
SDT-CGRA	5193865.40 (100%)	0.841 (100%)	29.443 (100%)
RC array	3381494.93 (65.11%)	0.524 (62.31%)	18.327 (62.25%)
SBUs	1449493.83 (27.91%)	0.199 (23.66%)	6.973 (23.68%)
Switch & Interfaces	362876.64 (6.99%)	0.118 (14.03%)	4.143 (14.07%)

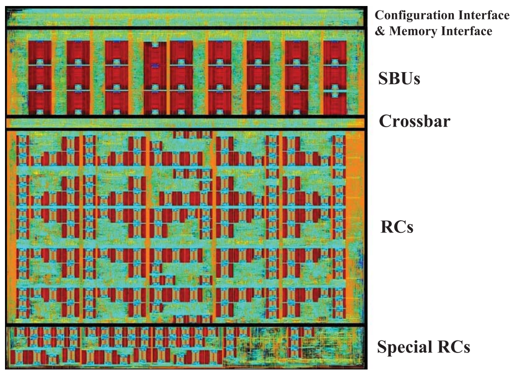


Fig. 16. Layout of SDT-CGRA (SMIC 55 nm).

and 119 ALUs, its peak performance can reach 92.3 GOP/s. The layout of the SDT-CGRA generated by IC compiler is shown in Fig. 16. It is worth to note that the input/output pads are not added, since the SDT-CGRA is not designed as an independent chip for acceleration. Instead, it is designed as a reconfigurable accelerator in a typical system-on-chip for object inference applications.

As for system power evaluation, we adopt an approximated evaluation method proposed in [25] to estimate the power

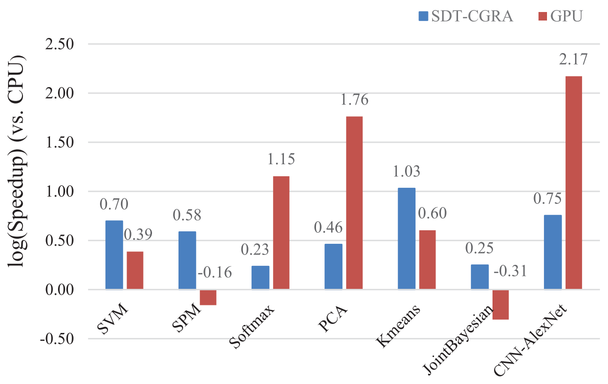


Fig. 17. Speedup of SDT-CGRA and GPU over CPU (the higher the better).

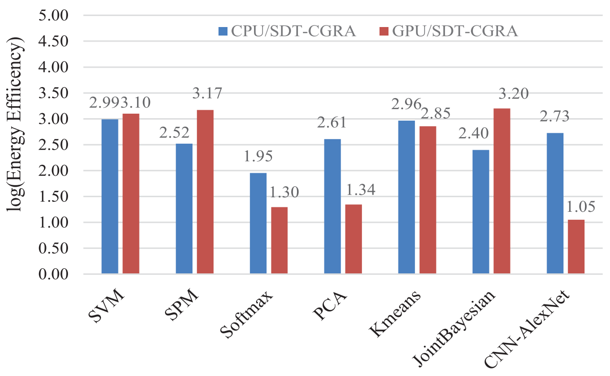


Fig. 18. Energy efficiency of SDT-CGRA compared with CPU and GPU.

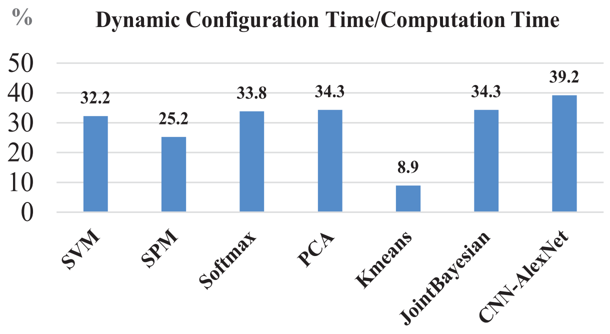


Fig. 19. Time cost in dynamic configuration over the total computation time.

consumption of the whole system

$$\text{Energy} = \text{Energy}_{\text{SDT-CGRA}} + \text{Energy}_{\text{off-chip}} \quad (4)$$

where $\text{Energy}_{\text{SDT-CGRA}}$ is the chip-only power and $\text{Energy}_{\text{off-chip}}$ is the power consumption in off-chip memory accesses. According to [41], the typical energy consumption of an off-chip memory (e.g., DDR3) is 70 pJ/bit. Consequently, we can estimate $\text{Energy}_{\text{off-chip}}$ according to the data assessment on the off-chip memory. For example, the estimated power consumption of AlexNet is shown in the following comparisons.

C. Comparisons With CPU and GPU

The algorithms in Table III are mapped on the proposed SDT-CGRA. We also implement these algorithms on both CPU and GPU. The CPU solution is based on the Intel E5-2637 (8 threads, 22-nm process) with the state-of-the-art Intel MKL library and Caffe [42] library, which are multi-threaded and widely used in linear algebra computing and deep learning applications. For the GPU solution, the algorithms are programmed with CUDA by cuBLAS [43] and Caffe library based on the Nvidia TitanX GPU (3584 CUDA cores, 16-nm process). In our evaluation approach, all the data are stored in the device memories before execution. In this case, the time cost in GPU implementations does not include the time of data transmission from host memory to the internal device memory. In SDT-CGRA, we assume that the data are stored in the off-chip memory with the bandwidth of 12.5 GB/s, which is a typical value of DDR3. Besides, we take the thermal design power of CPU (80 W) and GPU (250 W) provided by vendors as the power consumption of these two devices. As we do not take the energy consumption of the off-chip memory in CPU and GPU into account, only $\text{Energy}_{\text{SDT-CGRA}}$ is used for fair comparison.

To evaluate the speedup, the CPU solution is selected as baseline for comparison. Fig. 17 shows the accelerations of different algorithms on SDT-CGRA and GPU *versus* CPU. It is clear that the SDT-CGRA is faster than CPU ($\log(\text{Speedup}) > 0$) for all algorithms. For heavyweight algorithms [including Softmax, PCA, and CNN (AlexNet)], whose computation complexities are much larger than other algorithms in our experimental setup, GPU outperforms both CPU and SDT-CGRA greatly. For the lightweight algorithms (including SVM, SPM, k-means, and Joint Bayesian), SDT-CGRA gets better speedup than GPU, and the CPU can even faster than GPU in SPM and Joint Bayesian.

However, in the case of energy efficiency, the SDT-CGRA outperforms the CPU and GPU in all algorithms listed in Table III. We select the SDT-CGRA as the baseline for comparison. Results in Fig. 18 show that the energy cost in SDT-CGRA is smaller than the CPU and GPU. For more specific, SDT-CGRA can achieve on average 343.8 times and 17.7 times energy efficiency for heavyweight algorithms (including Softmax, PCA, and CNN), and 621.0 times and 1261.8 times energy efficiency for lightweight algorithms (including SVM, SPM, k-means, and Joint Bayesian) when compared with CPU and GPU.

As the SDT-CGRA is designed based on the proposed dual-track programming model, the overheads of static configuration on RCs are small enough to be neglected in the selected algorithms. On the contrary, the time cost in dynamic configuration, which is used to schedule data streams, is much higher. Fig. 19 shows the time cost in dynamic configuration over the total time cost in computing for a specific algorithm. As we have shown in Fig. 13(b), the time cost in fetching dynamic configuration contexts is mainly hidden by the time cost in computation. As a result, dynamic configuration has little effect on the overall processing performance.

D. Comparisons With FPGA and ASIC

Several representative highly customized implementations of CNN on FPGA and application-specific integrated

TABLE VI
COMPARISON WITH FPGA AND ASIC ACCELERATORS

	FPGA'15 [44]	FPL'16 [45]	JSSC'17 [9]	SDT-CGRA
Process	28nm	28nm	65nm	55nm
AlexNet Layer	Time (ms)			
Conv1	7.67	<2.56	5.23	4.23
Conv2	5.35	<2.56	10.48	8.21
Conv3	3.79	<2.56	5.9	4.82
Conv4	2.88	<2.56	4.6	3.62
Conv5	1.93	<2.56	2.63	2.36
FC6	–	<2.56	–	3.31
FC7	–	<2.56	–	1.47
Softmax	–	<2.56	–	0.36
Total	21.62	2.56	28.84	28.38
Frequency (MHz)	100	156	200	450
GOPS	61.62	565.94	46.2	77.4
System Power(W)	18.61	30.2	0.577	1.526
Energy (mJ)	c[1]: 402.3 a[2]: –	c : – a : 77.3	c : 16.6 a : –	c : 35.6 a : 43.3
Energy Efficency	c : 1× a : –	c : – a : 1×	c : 24.2× a : –	c : 11.3× a : 1.78×
GOPS/W	3.31	18.7	80.07	50.78
Frame/W	2.49	12.9	60.09	28.20

[1] c: contains convolutional layers only
[2] a: contains all layers

circuit (ASIC) are adopted for comparison. In [44], the acceleration of five convolutional layers of AlexNet on a Xilinx VC707 FPGA board is reported. In [45], a fully pipelined architecture (each layer is one pipeline stage) is proposed to accelerate all the layers of AlexNet on FPGA. In [9], an ASIC designed for CNN acceleration only is presented. The results are shown in Table VI, where the power consumption refers to the system power that includes the power dissipation in the off-chip memory. It is worth noting that only the results of convolutional layers are reported in [9] and [44]. As a result, we calculate the energy consumption and the energy efficiency in two different ways: convolutional layer only and all layers. From Table VI, we can see that SDT-CGRA achieves better results than FPGA in energy consumption, GOPS/W, and frames/W. Specifically, SDT-CGRA is 1.78 times better than the state-of-the-art acceleration of AlexNet in energy efficiency. As an ASIC implementation, [9] achieves a better energy efficiency than SDT-CGRA, due to its highly customized (for CNN) architecture and memory system. However, SDT-CGRA is a flexible architecture that is not only designed to support CNN but a wide range of algorithms.

E. Comparison With CGRA Implementations

EMAX [22] is proposed to accelerate CNN. In [22], OPs per memory bandwidth is adopted as the criteria to evaluate their architecture. And only the mapping results of the second convolutional layer of AlexNet are provided. According to [22], the number of OPs per memory bandwidth in EMAX is about 6. In SDT-CGRA, the whole architecture can reach 78.75 GOP/s when mapped with the second convolutional layer of AlexNet, with the requirement of 4.5-GB/s off-chip memory bandwidth. As a result, the number of OPs per memory bandwidth in SDT-CGRA can reach 17.5, which is almost three times of EMAX.

M-CGRA [23] is a CGRA architecture that is designed for CNN acceleration. For comparison purpose, we list the mapping results of AlexNet on M-CGRA and SDT-CGRA in Table VII. The power or energy consumption is not available in [23]. From Table VII, we can see that SDT-CGRA can achieve 13.4 times speedup compared with M-CGRA.

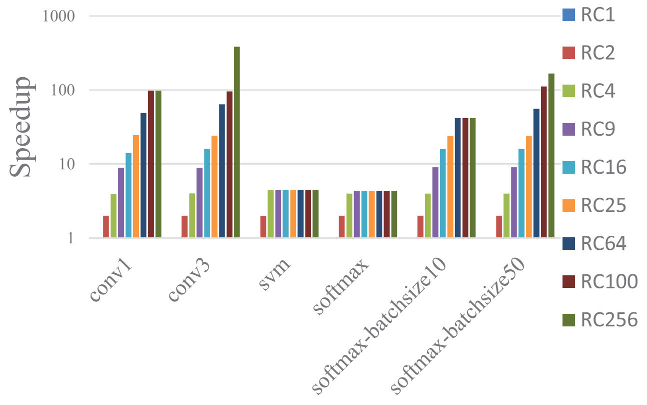


Fig. 20. Scalability of different computing patterns with a different number of RC. The *conv1* refers to the first convolutional layer of AlexNet. The *softmax-batchsize10* refers to the *softmax* with the batch size of 10. RC256 means that the number of RC is 256. The rests are similar.

F. Performance Scalability

The performance scalability of the SDT-CGRA is explored using several computing patterns. According to [46] and [47], the number of OPs per word (denoted by V_a) of a given algorithm determines its upper bound of computing performance. As a result, different computing patterns that have different values of V_a are selected for this paper, including: 1) convolution in the convolutional layer: $V_a \approx 2 \times k^2 \times O$ $(O \times k^2 \ll N^2)$ or $V_a \approx 2 \times N^2 (O \times k^2 \gg N^2)$, where k is the size of the convolution kernel, N is the size of input feature map, and O is the number of output feature map; 2) vector–vector multiplication: $V_a = 1$; 3) vector-matrix multiplication: $V_a = 2N/(N-1)$, where N is the size of vector; and 4) matrix–matrix multiplication: $V_a = N$, where N is the size of matrix.

The bandwidth of the off-chip memory for the SDT-CGRA is assumed to be 12.5 GB/s. With this premise, the performance scalability of SDT-CGRA is estimated with different numbers of RCs. The results are shown in Fig. 20, where the performance improvements of *conv1* (the first convolutional layer in AlexNet) and *conv3* (the third convolutional layer in AlexNet) are nearly linear with the increase of the number of RCs. For *conv1*, the performance under RC100 and RC256 is nearly the same. The reason is that, when the number of output feature maps O is less than the number of RCs,

TABLE VII
COMPARISON BETWEEN M-CGRA AND SDT-CGRA

AlexNet Layer	Conv1	Conv2	Conv3	Conv4	Conv5	FC6	FC7	FC8	Total		GOPS	System Power (W)
	Time (ms)								Time (ms)	Speedup		
M-CGRA [23]	80.72	98.30	106.17	19.91	13.27	55.38	5.24	1.31	380.3	1×	86.37	–
SDT-CGRA	4.23	8.21	4.82	3.62	2.36	3.31	1.47	0.36	28.38	13.4×	77.49	1.526

the performance will saturate. As for *svm*, which involves vector–vector multiplication, the performance relies heavily on the off-chip memory bandwidth. The *softmax*, an algorithm that contains a vector-matrix multiplication process, also suffers from the same problem. However, when we adopt the batch processing technique (the vector-matrix multiplication is converted to matrix–matrix multiplication), the problem can be alleviated. For example, when the batch size increases from 1 to 50, the performance will improve correspondingly with the increase in the number of RCs. It is worth to note that batch processing is only useful for applications that are not sensitive to latency.

VII. CONCLUSION

This paper proposes a stream processing, dual-track programming CGRA, which targets algorithms in the object inference flow. The proposed SDT-CGRA is implemented using the SMIC 55-nm standard cell technology with a footprint of 5.19 mm^2. When running at 450 MHz over seven typical algorithms, the average chip-only power consumption of SDT-CGRA is 0.84 W. When compared with CPU and GPU, the SDT-CGRA can gain on average 343.8 times and 17.7 times energy efficiency for heavyweight algorithms, and 621.0 times and 1261.8 times energy efficiency for lightweight algorithms, respectively. Although the SDT-CGRA does not gain competitive energy efficiency compared with ASIC solution for specific algorithms, SDT-CGRA in a 55-nm transistor technology can achieve 1.78 times improvement in energy efficiency compared with the state-of-the-art solution of AlexNet on FPGA in a 28-nm transistor technology. When compared with the CGRA approach, SDT-CGRA is three times better than EMAX in terms of OPs per memory bandwidth and 13 times of M-CGRA in terms of speedup. Current and future work includes extending the proposed approach for other applications and automating the development of the associated compilation and debugging tools.

REFERENCES

[1] X. Fan, H. Li, W. Cao, and L. Wang, “DT-CGRA: Dual-track coarse-grained reconfigurable architecture for stream applications,” in *Proc. Int. Conf. Field Programm. Logic Appl.*, Aug./Sep. 2016, pp. 78–86.

[2] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, and G. Penn, “Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition,” in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2012, pp. 4277–4280.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[4] R. Ranganath, A. Perotte, N. Elhadad, and D. Blei. (2016). “Deep survival analysis.” [Online]. Available: https://arxiv.org/abs/1608.02158

[5] N. Jouppi. *Google Supercharges Machine Learning Tasks With TPU Custom Chip*. Accessed: Nov. 26, 2016. [Online]. Available: https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html

[6] N. P. Jouppi *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Proc. ACM/IEEE 44nd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2017, pp. 1–12.

[7] Y. Chen *et al.*, “DaDianNao: A machine-learning supercomputer,” in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.

[8] Z. Du *et al.*, “ShiDianNao: Shifting vision processing closer to the sensor,” in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2015, pp. 92–104.

[9] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[10] S. Liu *et al.*, “Cambricon: An instruction set architecture for neural networks,” in *Proc. ACM/IEEE 43nd Annu. Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 393–405.

[11] M. J. Flynn and W. Luk, *Computer System Design: System-on-Chip*. Hoboken, NJ, USA: Wiley, 2011.

[12] G. Ansaloni, P. Bonzini, and L. Pozzi, “EGRA: A coarse grained reconfigurable architectural template,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 6, pp. 1062–1074, Jun. 2011.

[13] J. Cong, H. Huang, C. Ma, B. Xiao, and P. Zhou, “A fully pipelined and dynamically composable architecture of CGRA,” in *Proc. IEEE 22nd Annu. Int. Symp. Field-Programm. Custom Comput. Mach.*, May 2014, pp. 9–16.

[14] J. Davila, A. de Torres, J. M. Sanchez, M. Sanchez-Elez, N. Bagherzadeh, and F. Rivera, “Design and implementation of a rendering algorithm in a SIMD reconfigurable architecture (MorphoSys),” in *Proc. Conf. Design, Autom. Test Eur., Designers’ Forum*, Mar. 2006, pp. 52–57.

[15] F. Bouwens, M. Berekovic, A. Kanstein, and G. Gaydadjiev, “Architectural exploration of the ADRES coarse-grained reconfigurable array,” in *Proc. 3rd Int. Workshop Reconfigurable Comput., Archit., Tools Appl. (ARC)*, Mangaratiba, Brazil, Mar. 2007, pp. 1–3.

[16] V. Govindaraju *et al.*, “DySER: Unifying functionality and parallelism specialization for energy-efficient computing,” *IEEE Micro*, vol. 32, no. 5, pp. 38–51, Sep. 2012.

[17] O. Atak and A. Atalar, “BilRC: An execution triggered coarse grained reconfigurable architecture,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 7, pp. 1285–1298, Jul. 2013.

[18] Y. Huang, P. Ienne, O. Temam, Y. Chen, and C. Wu, “Elastic CGRAs,” in *Proc. ACM/SIGDA Int. Symp. Field Programm. Gate Arrays*, 2013, pp. 171–180.

[19] J. Cortadella, M. Kishinevsky, and B. Grundmann, “Synthesis of synchronous elastic architectures,” in *Proc. 43rd Annu. Design Autom. Conf.*, 2006, pp. 657–662.

[20] L. Liu *et al.*, “An energy-efficient coarse-grained reconfigurable processing unit for multiple-standard video decoding,” *IEEE Trans. Multimedia*, vol. 17, no. 10, pp. 1706–1720, Oct. 2015.

[21] S. Cadambi, A. Majumdar, M. Becchi, S. Chakradhar, and H. P. Graf, “A programmable parallel accelerator for learning and classification,” in *Proc. 19th Int. Conf. Parallel Archit. Compilation Technol.*, 2010, pp. 273–284.

[22] M. Tanomoto, S. Takamaeda-Yamazaki, J. Yao, and Y. Nakashima, “A CGRA-based approach for accelerating convolutional neural networks,” in *Proc. IEEE Int. Symp. Embedded Multicore/Many-Core Syst.-Chip*, Sep. 2015, pp. 73–80.

[23] K. Ando, S. Takamaeda-Yamazaki, M. Ikebe, T. Asai, and M. Motomura, “A multithreaded CGRA for convolutional neural network processing,” *Circuits Syst.*, vol. 8, no. 6, pp. 149–170, 2017.

[24] D. Shin, J. Lee, J. Lee, and H.-J. Yoo, “14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks,” in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 240–241.

[25] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 8, pp. 2220–2233, Aug. 2017.

[26] S. Yin *et al.*, "A 1.06-to-5.09 TOPS/W reconfigurable hybrid-neural-network processor for deep learning applications," in *Proc. Symp. VLSI Circuits*, Jun. 2017, pp. C26–C27.

[27] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Sep. 2009.

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. CVPR*, Jun. 2009, pp. 248–255.

[29] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2006, pp. 2169–2178.

[30] D. Chen, X. Cao, L. Wang, F. Wen, and J. Sun, "Bayesian face revisited: A joint formulation," in *Proc. Eur. Conf. Comput. Vis.*, 2012, pp. 566–579.

[31] P. Felzenszwalb, D. Mcallester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2008, pp. 1–8.

[32] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," *Comput. Vis. Image Understand.*, vol. 110, no. 3, pp. 404–417, 2006.

[33] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[34] K. Simonyan and A. Zisserman. (2014). "Very deep convolutional networks for large-scale image recognition." [Online]. Available: https://arxiv.org/abs/1409.1556

[35] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[36] L. Wan, C. Dong, and D. Chen, "A coarse-grained reconfigurable architecture with compilation for high performance," *Int. J. Reconfigurable Comput.*, vol. 3, Jan. 2012, Art. no. 3.

[37] F. Bistouni and M. Jahanshahi, "Scalable crossbar network: A non-blocking interconnection network for large-scale systems," *J. Supercomput.*, vol. 71, no. 2, pp. 697–728, 2015.

[38] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, "Convolution engine: Balancing efficiency & flexibility in specialized computing," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 24–35, 2013.

[39] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[40] C. Lomont, "Fast inverse square root," *Tech-315 Nical Rep.*, vol. 32, 2003.

[41] D. SDRAM. *JESD79-3F*. Accessed: Nov. 26, 2016. [Online]. Available: http://www.jedec.org/standards-documents/docs/jesd-79-3d.

[42] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.

[43] Nvidia. *cuBLAS*. Accessed: Nov. 26, 2016. [Online]. Available: https://developer.nvidia.com/cublas

[44] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programm. Gate Arrays*, 2015, pp. 161–170.

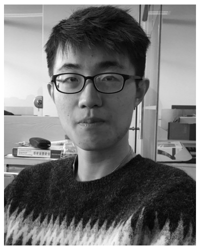
[45] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," in *Proc. Int. Conf. Field Programm. Logic Appl.*, 2016, pp. 69–77.

[46] R. Gruber, P. Volgers, A. De Vita, M. Stengel, and T.-M. Tran, "Parameterisation to tailor commodity clusters to applications," *Future Generat. Comput. Syst.*, vol. 19, no. 1, pp. 111–120, 2003.

[47] S. Rousseau, D. Hubaux, P. Guisset, and J.-D. Legat, "A high performance FPGA-based accelerator for BLAS library implementation," in *Proc. 3rd Annu. Reconfigurable Syst. Summer Inst.*, 2007, pp. 1–10.

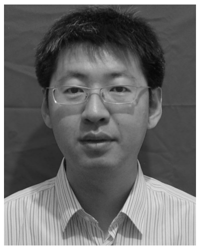
Xitian Fan received the B.S. degree from the School of Physics and Engineering, Sun Yat-sen University, Guangzhou, China, in 2012 and the M.S. degree from the School of Microelectronics, Fudan University, Shanghai, China, in 2014, where he is currently working toward the Ph.D. degree at the School of Information Science and Technology.

His current research interests include the reconfigurable computing, computer architecture, and machine learning acceleration on field-programmable gate array.



Di Wu received the B.S. degree from the School of Information Science and Technology, Fudan University, Shanghai, China, in 2015, where he is currently working toward the Ph.D. degree.

His current research interests include computer architecture and the development of reconfigurable systems.



Wei Cao (M'14) received the B.S. and M.S. degrees in from Heilongjiang University, Harbin, China, in 1996 and 2000, respectively, and the Ph.D. degree from the Harbin Institute of Technology, Harbin, China, in 2006.

Since 2009, he has been an Assistant Researcher with the State Key Laboratory of ASIC and System, Fudan University, Shanghai, China. His current research interests include reconfigurable computing, and field-programmable gate arrays architectures and VLSI architectures for digital video and image processing.

Wayne Luk (F'09) received the M.A., M.Sc., and D.Phil. degrees in engineering and computing science from Oxford University, Oxford, U.K.

He founded and leads the Computer Systems Section and the Custom Computing Group, Department of Computing at Imperial College London. He was a Visiting Professor at Stanford University, Stanford, CA, USA, and Queens University Belfast, Belfast, U.K. He is currently a Professor of Computer Engineering at the Imperial College London, London, U.K. He has been an author or editor for six books and four special journal issues.

Dr. Luk is a member of the Program Committee of many international conferences, such as the IEEE International Symposium on Field-Programmable Custom Computing Machines, the International Conference on Field-Programmable Logic and Application (FPL), and the International Conference on Field-Programmable Technology (FPT). He is a Fellow of the Royal Academy of Engineering and the British Computer Society Ltd. He had 15 papers that received awards from various conferences, such as the IEEE International Conference on Application-specific Systems, Architectures and Processors, FPL, FPT, the International Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation, X Southern Programmable Logic Conference, and the European Regional Science Association. He also received the Research Excellence Award from the Imperial College London in 2006. He was a founding Editor-in-Chief of the *ACM Transactions on Reconfigurable Technology and Systems*.

Lingli Wang (M'99) received the M.S. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 1997 and the Ph.D. degree in electrical engineering from Edinburgh Napier University, Edinburgh, U.K., in 2001.

He was with the Altera European Technology Center, High Wycombe, Buckinghamshire, U.K., for four years. In 2005, he joined Fudan University, Shanghai, China, where he is currently a Full Professor at the State Key Laboratory of ASIC and System, School of Microelectronics. His current research interests include logic synthesis, reconfigurable computing, and quantum computing.