Stream Processing Dual Track CGRA for Object Inference

Xitian Fan, Di Wu, Wei Cao, 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 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 SDT-CGRA is presented as an implementation prototype. To evaluate the performance, the SDT-CGRA is realized in Verilog HDL and implemented in SMIC 55nm process, with the footprint of 5.19 mm² at 450 MHz. Seven object inference algorithms including CNN, k-means, PCA, SPM, linear-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 to the Intel Xeon E5-2637 CPU and the Nvidia TitanX GPU. When compared to the state-of-the-art solutions of AlexNet on FPGA and CGRA, the proposed SDT-CGRA can achieve a 1.78 times increase in energy efficiency and a 13 times speedup respectively.

Index Terms—coarse-grained reconfigurable architecture, domain-specific computing, machine learning, deep learning, object inference.

I. INTRODUCTION

With the breakthrough of deep learning technology in speech applications [2], computer vision [3] and other tasks in artificial intelligence (AI) [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 to the traditional approaches [5], [6]. In academia, deep learning accelerator, such as DaDianNao [7], ShiDianNao [8], EyeriSS [9], Cambricon [10], etc., have shown impressive improvements in energy-efficiency compared to general purpose processors and GPUs. However, application specific accelerators (e.g. DaDianNao, ShiDianNao, EyeriSS) are often tailored to specific algorithms (e.g. Convolutional Neural Network, CNN). The hardwired 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 machine learning, especially the object inference flow.

The increasing demand to process data streams more efficiently for data-centric applications for 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 an algorithm can be divided into multiple computing kernels while each kernel computes in the stream manner. The second is fixed kernel-level operations, which means 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.

To design a CGRA that caters to these characteristics, we first employ a cluster of processing elements (e.g. ALU, Multiplier) as an elementary reconfiguration cell (RC) for kernel-level operations. 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 operations can be reduced and hence the computation can be speeded up. This approach is similar to EGRA [12] and FPCA [13], which have proved the clustered RC array can outperform traditional CGRAs that can only map one computing operation in each cell. Secondly, stream processing is adopted as programming paradigm. Due to the characteristic of the fixed kernel-level operations, 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 operations, 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 operations (OPs) from context memory for their RCs in every execution cycle, as illustrated in Fig.2(a). According to the characteristic of the fixed kernel-level operations 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. Thirdly, 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.

- A novel CGRA-based approach to accelerate algorithms in a general object inference flow is provided. This approach has three aspects. Firstly, it adopts stream processing, such that each kernel-level RC computes in stream manner. Secondly, it employs 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.

- A CGRA implementation prototype, called Stream Dual-Track CGRA (SDT-CGRA), is provided to realize the above 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 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 the 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 (SIMD) 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 operation make 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 operations 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 operations are suitable for specialization.

Kernel-level operations in stream processing require kernel-based processing units in order to compute efficiently. EGRA
[12], an expression-level granularity CGRA framework, has showed that expression-level or kernel-level CGRA fabric can outperform traditional CGRA approaches. In this paper, reconfigurable block in kernel-level granularity is employed 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.

BiRRC [17] and Elastic CGRA [18] can be categorized into another CGRA template. Both of them are similar to commercial FPGA in architecture arrangement and configuration manner. For BiRRC, 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 our work was presented in [1]. This paper further optimizes the memory of RC and SDT-CGRA. More specifically, the number of ALUs, 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, BiRRC, Elastic CGRA mainly target at the acceleration of FFT, DCT/IDCT, etc. 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) are 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 algorithms specific approaches like Eyeriss [9], DNPU [24], DNA [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, convolutional neural network (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.

Generally, the detection of objects requires to process images or feature maps by scanning operations within specific processing scopes in multiple scales. Each scanning operation 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 illustrated in Fig. 4(a). As for multi-scale detection, general methods include classifiers running on the pyramid of images/feature maps (e.g. DPM [31]), pyramid of filters running on the feature maps (e.g. SURF [32]) and pyramid of referenced bounding boxes on the final regression functions (e.g. Faster R-CNN [33]). For simplicity, multi-scale detection can be regarded as multi-layer
scanning operation, as illustrated 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 corresponds to the fixed kernel-level operations. In this case, the input 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 multi-scale/multi-layer operations have pyramid input images/feature maps, or pyramid output of intermediate results, which means 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

- stream processing,
- fixed kernel-level operations,
- ample storage.

In our selected representative algorithms, kernel-level operations 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 fully connected layer (contains matrix multiplication pattern in Table II) and softmax layer (contains matrix multiplication and interpolation pattern in Table II), can also be organized as kernel-level operations 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.

- Design kernel-level granularity RCs in stream manner.
- Increase the size of intermediate storage.
- Employ 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 dual-track programming model in this paper, explains how the architecture works. We will discuss these two parts in detail below.

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 operations. As a result, the elementary RC should be based on multiplier-ALU (MUL-ALU for short) units. An approach such as placing multipliers and ALUs in different part, e.g. BiRC [17], would increase the routing resource requirements and even lead to the placement or routing conflict. As a result, the tightly couple of multiplier and ALU for our applications is a proper approach.

On the other hand, extra operations are required to support further additions, accumulations or logic operations in convolution, matrix multiplication, distance calculation, etc. If single MUL-ALU unit is employed as the elementary RC, the extra additions, accumulations or logic operations will lead to low utilization ratio of multipliers in MUL-ALU units. As a result, combining MUL-ALU units and extra ALUs as 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 $3 \times 3$ (denoted as conv$_{3x3}$) ranks number one in AlexNet and VGG-16, while GoogleNet has the most number of the $1 \times 1$ convolution kernels (denoted as conv$_{1x1}$). However, from the perspective of computation, the $3 \times 3$ convolution
Kernel-level iteration of the convolution kernel still ranks number one in three CNN architectures (see Fig. 5(b)). In VGG-16, the computation ratio of conv_3x3 in all the convolutional layers is even up to 100%. As a result, efficient support for 3 x 3 convolution kernel can be regarded as the foundation to design a 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 illustrated in Fig. 6(a). In order to introduce the static and dynamic configurations clearly, we can re-organize 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 VLIW format, while the RC array is configured statically according to the computing kernels. The CGRA architecture based on 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 SBU or from the SBU to the RC array, 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 SBU starts to generate addresses to issue load or store operations (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 operations.

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 operation. For simplicity, we assume the input image Imap contains two rows of data, denoted as L1 and L2. The size of the convolution kernel is 1 x 3. Based on the dual-track model, the convolution operation 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 concurrent operations, which are used to load the input data and store back the results. For example, in Instruction 1, L1 is read out from SBU and then sent to RC1. At the same time, the output stream result R1 is stored back to SBU as soon as it is available. Instruction 2 performs the same operations except that the input data stream and the output stream are different. In this method, the 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 consume one clock cycle to finish the configuration with the bandwidth requirement of BW on RC1. As for the approach that requires to load OPs in every execution cycle, the total bandwidth requirement in loading OPs is 14 x BW (The Imap in Fig. 7(b) has been cropped for clarity).
14 elements). As for our static configuration approach, the bandwidth requirement is only BW since we only need to configure the RC1 in one time. From this point of 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. The 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 operations 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 IRC and PRC in Fig.8). Special RCs are used to support some special operations, such as power function (corresponding to PRC) and transcendental function that can be approximated by interpolation (corresponding to IRC). Since such operations account for small computation generally, only several special RC units are provided. Details of these units are introduced in the 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 horizontal direction. As a result, the larger computing kernel that exceed the computing volume of one RC can be realized by several RCs, and help 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 vertical direction. This type of direct data transmission method is inspired by the FDR-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 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 memory DMA 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.

![Fig. 8. The typical acceleration system consists of an SDT-CGRA architecture, an off-chip memory and a host processor.](image)

![Fig. 9. The micro architecture of RC from the data path perspective.](image)

![Fig. 10. Three different work style of MUL-ALU units. The \( z^n \) means to delay \( n \) clock cycles of the data.](image)
B. RC Architecture

1) Detailed Architecture: The guidelines in Section III-B shows 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 SDF-CGRA. Three MUL-ALU units can be configured to execute multiply-accumulate operations, 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 operations to reduce the bandwidth of output interface. This idea follows [38] where it is called “map-reduce” structure. In our architecture, three MUL-ALU units belong to the “map” part, while the remaining ALU belongs to “reduce” part (see Fig. 9). The “map” part can be used to execute concurrent operations 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 to 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 operation intensity per input data, which can help to improve computation performance according to the roofline model [39].

To implement stream processing, the input and output stream interface in Fig. 3 are realized with two types of local memories (FIFOs and 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 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 are 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 (FSMs) 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, clear signals for ALUs when they are configured to be accumulators, ready and valid signals in interconnection channels for adjacent RCs, etc.

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 = *(int*)x;
3. float j = (1.0 - p) * 1064975338 + (p * i); // 1064975338 is called magic number
4. i = int(j);
5. x = *(float*)i;
6. y = (1.0 - p) * x + p * pow(x, (p - 1)/p) * xorig;

C. Special RC Design

Two types of special RCs, PRC and IRC, are developed to support the power functions and piecewise functions respectively. Consider first PRC, designed to calculate \( x^{1/2} \), \( x^{-1/2} \) and \( x^{-1} \) based on the fast inverse square root algorithm [40] shown in Algorithm 1. Many multiplications in Algorithm 1 can be simplified to be shifting or addition operations, while the \( \text{pow}(x, (p - 1)/p) \) can be calculated by multiplications when \( p \) set to be 1/2, -1/2 or -1. Due to the requirement of 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 the 6-th line 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.

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

Specifically, the input data \( x \) is compared with the boundary values of each interval from \( x_0 \) to \( x_{N-1} \) in parallel to generate an address for look up tables. Then the coefficients of \( a_i \) and \( b_i \) from look up 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 SDF-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 (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) The Interconnections Among RCs: A simple example is used to introduce the functionality of the Reverse-S interconnections among RCs. Assuming that five multiplier-ALUs are required to implement an expression \( E \), three multiplier-ALUs in the first RC and two multiplier-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) illustrates such an approach, where five RCs in two adjacent rows are configured to calculate three such expressions in parallel.

Fig. 12(b) illustrates 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 from 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) The 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 operations of 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 \( k \)-th SBU issues several write operations to the \( j \)-th RC in the \( i \)-th row (denoted as \( RC_{ij} \) in Fig. 13(a)). In the first VLIW cycle, the \( k \)-th 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 \( k \)-th 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 illustrated 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 operation 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 operation and 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 operations in the convolutional layers can support various sizes of convolution kernels with arbitrary strides. Without loss of generality, suppose that the mapped kernel is \( 5 \times 5 \), the stride is \( 2 \times 2 \), and the width of the 2D input feature map is \( N \). The number of multiplier-ALUs that are required to map such a convolution kernel is determined by \( \lfloor 5/2 \rfloor \times \lfloor 5/2 \rfloor \). 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 convolution kernel is extended to be \( 6 \times 6 \) 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.
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 $\omega$ from 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 a certain number of input features, e.g. 100, are batched together to construct a larger matrix, such as $F_{100 \times 9216}$. In this way, the weights can be reused 100 times so that the time cost to load new weights into the SRAM blocks can be hidden by the computing time. Due to the capacity limitation of each SRAM block, the weight matrix have to be divided into several smaller sub-matrices so that each one can be loaded into one SRAM block. In order to adopt double-buffer strategy, the dimension of each sub-matrix should not exceed half of the size of a 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 $F_{100 \times 9216}$ is divided into 100 $72$ blocks. Each block is denoted as $F_{i,j}$, 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 $t_1: \omega_{129 \times 75}$ and the other is $t_2: \omega_{m,n \times 46}$, where $m$ and $n$ are the indices. After the partition, equation (2) can be expressed as

$$O_{100 \times 4096} = F_{1 \times 9216} \times \omega_{9216 \times 4096}$$

To illustrate the computing process of equation (3) on the SDT-CGRA, we take the multiplication process of $F_{100 \times 9216}$ with $\omega_{1,0}$ as an example. The computing process mapped into the SDT-CGRA is shown in Fig.15. Firstly, 75 columns of $\omega_{1,0}$ are loaded into 75 SRAMs in the RC array. Then the sub-matrices $F_{0,0}, F_{1,0}, ..., F_{99,0}$ in the first column of $F_{100 \times 9216}$ 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 $\omega_{1,0}$ and the second column of $F_{100 \times 9216}$. 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 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 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 operations into a library. To map a given task into the SDT-CGRA, it is programmed with the provided APIs of the library.

B. Implementation Details

The proposed SDT-CGRA contains $5 \times 5$ (25) RCs, 5 special RCs, 54 KByte global SRAM (27 SBU) and 54.6 KByte local memory (including FIFOs and SRAMs). The detailed information is shown in Table IV, where all the computing units are 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 Synopsys Design Compiler and IC Compiler based on the 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 7 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 the architecture can run at 450 MHz. Since the SDT-CGRA contains 86 multipliers 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 IO pads is not added in the layout, 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 (SoC) for object inference applications.

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

$$\text{Energy} = \text{Energy}_{\text{SDT-CGRA}} + \text{Energy}_{\text{off-chip}}$$  

(4)

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

---

**TABLE III**

**TYPICAL IMPLEMENTATIONS, APPLICATIONS AND PROBLEM SIZES OF THE SELECTIVE ALGORITHMS**

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

---

**TABLE IV**

**DETAILED INFORMATION OF EACH UNIT**

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

---

**TABLE V**

**CHARACTERISTICS OF THE LAYOUT OF SDT-CGRA AND THE AVERAGE POWER DISSIPATION AND TOTAL ENERGY CONSUMPTIONS OVER 7 BENCHMARKS**

<table>
<thead>
<tr>
<th>Component</th>
<th>Area ($\mu$m$^2$)</th>
<th>Chip-only Power (W)</th>
<th>Chip-only Energy (mJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDT-CGRA</td>
<td>5193865.40 (100%)</td>
<td>0.841 (100%)</td>
<td>29.443 (100%)</td>
</tr>
<tr>
<td>RC array</td>
<td>3381494.93 (65.11%)</td>
<td>0.524 (62.31%)</td>
<td>18.327 (62.25%)</td>
</tr>
<tr>
<td>SBUs</td>
<td>1449403.83 (27.91%)</td>
<td>0.199 (23.66%)</td>
<td>6.973 (23.68%)</td>
</tr>
<tr>
<td>Switch &amp; Interfaces</td>
<td>362876.64 (6.99%)</td>
<td>0.118 (14.03%)</td>
<td>4.143 (14.07%)</td>
</tr>
</tbody>
</table>
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, 22nm process) with the state-of-the-art Intel MKL library and Caffe [42] library, which are multithreaded 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, 16nm 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 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 (TDP) 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 off-chip memory in CPU and GPU into account, only the Energy\_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 vs. CPU. It is clear that the SDT-CGRA is faster than CPU (log(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 outperform both CPU and SDT-CGRA greatly. For the lightweight algorithms (including SVM, SPM, k-means, 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 are 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, Joint Bayesian) when compared to 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 computation 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 ASIC are adopted for comparison. In [44], the acceleration of five convolutional layers of AlexNet on 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 off-chip memory. It is worth noting that only the results of convolutional layers are reported in [44] and [9]. As a result, we calculate the energy consumption and the energy efficiency in two different way: convolutional layer only and all layers. From Table VI we can see that SDT-CGRA achieves better results than FPGA in energy consumption, GOPS per watt and frame per watt. 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 convolutional neural network. In [22], operations 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 operations per memory bandwidth in EMAX is about 6. In SDT-CGRA, the whole architecture can reach 78.75 GOPs 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 operations per memory bandwidth in SDT-CGRA can reach 17.5, which is almost 3 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 SD-CGRA can achieve 13.4 times speedup compared to M-CGRA.

**F. Performance Scalability**

The performance scalability of the SDT-CGRA is explored using several computing patterns. According to [46], [47], the number of operations 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 study, including (1) convolution in convolutional layer: $V_a \approx 2 \times k^2 \times O \quad (O \times k^2 \ll N^2)$ or $V_a \approx 2 \times N^2 \quad (O \times k^2 \gg N^2)$, where $k$ is the size of the convolution kernel, $N$ is the size of input feature map, $O$ 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; (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 the 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, the performance will saturate. As for $svm$, which involves vector-vector multiplication, the performance relies heavily on 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 coarse-grained reconfigurable architecture which targets algorithms in the object inference flow. The proposed SDT-CGRA is implemented using the SMIC 55nm standard cell technology with a footprint of 5.19 mm$^2$. When running at 450 MHz over 7 typical algorithms, the average chip-only power consumption of SDT-CGRA is 0.84 W. When compared to 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 to ASIC solution for specific algorithms, SDT-CGRA in 55nm transistor technology can achieve 1.78 times improvement in energy efficiency compared to the state-of-the-art solution of AlexNet on FPGA in 28nm transistor technology. When compared to the CGRA approach, SDT-CGRA is 3 times better than EMAX in terms of operations 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


<table>
<thead>
<tr>
<th>AlexNet Layer</th>
<th>Conv1</th>
<th>Conv2</th>
<th>Conv3</th>
<th>Conv4</th>
<th>Conv5</th>
<th>FC6</th>
<th>FC7</th>
<th>FC8</th>
<th>Total</th>
<th>GOPS</th>
<th>System Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>M-CGRA [23]</strong></td>
<td>80.72</td>
<td>98.30</td>
<td>106.17</td>
<td>19.91</td>
<td>13.27</td>
<td>55.38</td>
<td>5.24</td>
<td>1.31</td>
<td>380.3</td>
<td>1×</td>
<td>86.37</td>
</tr>
<tr>
<td><strong>SDT-CGRA</strong></td>
<td>4.23</td>
<td>8.21</td>
<td>4.82</td>
<td>3.62</td>
<td>2.36</td>
<td>3.31</td>
<td>1.47</td>
<td>0.36</td>
<td>28.38</td>
<td>13.4×</td>
<td>77.49</td>
</tr>
</tbody>
</table>


