DT-CGRA: Dual-Track Coarse-Grained Reconfigurable Architecture for Stream Applications

Xitian Fan, Huimin Li, Wei Cao*, Lingli Wang
State Key Laboratory of ASIC and System, Fudan University
No. 825 Zhangheng Road, Shanghai, 201203, China
*caow@fudan.edu.cn

Abstract—This paper presents a new type of coarse-grained reconfigurable architecture (CGRA) for the object inference domain in machine learning. The proposed CGRA is optimized for stream processing and a correspondent programming model called dual-track model is proposed. The CGRA is realized in Verilog HDL and implemented in SMIC 55 nm process, with the footprint of 3.79 mm² and consuming 1.79 W at 500 MHz. To evaluate the performance, eight machine-learning algorithms including HOG, CNN, k-means, PCA, SPM, linear-SVM, Softmax and Joint-Bayesian are selected as benchmarks. These algorithms cover a general machine learning flow in object inference domain: feature extraction, feature selection and inference. The experimental results show that the proposed CGRA can gain 1443x average energy efficiency comparing to the Intel i7-3770 CPU and 7.82x energy efficiency comparing to a high performance FPGA solution [19].

Keywords—coarse-grained reconfigurable architecture; domain-specific computing; machine learning; object inference

I. INTRODUCTION

Energy-efficiency and flexibility are two analytical perspectives in architecture design community. As Dennard scaling becomes near to the end, there is a strong demand to develop energy-efficient architecture. On one hand, the increasing divergence between energy-efficiency and transistor-density at device level [1] makes it harder to improve the energy-efficiency at architecture level for the general-purpose computing system. On the other hand, when the computing-intensive and data intensive applications are prevalent in various domains, the challenges in terms of energy consumption and space requirement of the general-purpose computing system become severe [2]. One emerging approach to deal with these problems is the domain-specific computing. As one of promising paradigms, coarse-grained reconfigurable architecture (CGRA) has been proved to outperform the general-purpose computing systems and the fine-grained reconfigurable architecture (e.g. FPGA) in specific domains [3], [4], [5], [6], [7]. One example about the image edge detection by Sobel operator reported in [6] has shown that 13.3x speedup and 57x energy reduction can be achieved compared to the dual-core ARM Cortex-A9 processor.

With the great achievements of deep learning in natural language processing, object inference and speech recognition [21], [13], [22], developing energy-efficient and flexible architectures for deep learning algorithms is one of the hotspots currently both in the industry and the academic community. In this context, we try to design a CGRA for the object inference domain. Comparing to the existing CGRAs running with traditional applications such as fast Fourier transform (FFT) and discrete cosine transform (DCT), the proposed architecture is optimized according to the representative algorithms from feature extraction, feature selection and inference in object detection flow. Besides, the proposed CGRA is designed to work in a stream style. And as a result, the reconfigurable cells (RCs) and the correspondent interconnections of the proposed CGRA are optimized for the stream processing.

The key contributions of this paper can be summarized as follows.

1. We propose a CGRA that aims for the object inference domain in machine learning. The RCs and the interconnections in the proposed CGRA are optimized for stream processing.

2. A dual-track programming model is proposed for the CGRA. The model contains two configuration steps, static configuration and dynamic configuration. The static configuration is to determine the functionalities of mapped RCs while the dynamic configuration is to manage the data streams.

3. A novel RC decomposition and combination strategy is proposed to provide the flexibility to map kernel functions onto RCs.

The rest of this paper is organized as follows. Section II review the previous works from the architecture perspective. Section III demonstrates the proposed dual-track programming model with a simple example. Section IV introduces the details of each component in the DT-CGRA. In Section V, two mapping examples are demonstrated to show mapping strategies. The experimental results are shown in Section VI. Finally we conclude the paper in Section VII.

II. RELATIVE WORK

From the architecture perspective, CGRAs proposed in the past decades are various. In [3], a word-level reconfigurable architecture called MorphSys has been proposed. The reconfigurable cells (RCs) of MorphSys are organized as a 2D mesh array and a global frame buffer is provided for caching input data, intermediate and final results. No local memory of each RC is available. The context configurations of the RC array are controlled in a SIMD-style by a powerful RISC processor. The MorphSys has the following drawbacks. Firstly, the lack of local memory increases the global frame buffer access conflicts. Secondly, the SIMD-style execution of all the RCs in the same row or column results in low resource utilization in some common computation patterns, e.g., the adder-tree computing pattern in stream applications. Another similar architecture called ADRES is proposed in [4]. Comparing to the MorphSys,
the ADRES is a tightly coupled architecture where the RC array is configured to be one execution branch of the VLIW processor. The advantage of this approach is the favorable compatibility with the existing compiler. The configuration contexts can be regarded as customized instructions of the VLIW processor. However, for stream applications, the ADRES is not efficient due to the limited operations per instruction reference.

Compared with MorphSys and ADRES, the RC architecture in MORE [5] is designed with a local memory block apart from an arithmetic block and a logic block. Connections among RCs are programmed statically so that overhead of context issuing is small for stream applications. However, there is no global memory for data caching between the off-chip memory and local memory blocks. This two-layer memory hierarchy may result in two disadvantages. Firstly, data transmission to the off-chip memory is the bottleneck when the RCs have to access the off-chip memory frequently. Secondly, it is not efficient for MORE to support stream locality. The streams that are used in different RCs have to be duplicated in different local memory blocks.

Another impressive architecture called FPCA is proposed in [6], where each RC is a cluster of PEs and local memories. The advantage of such an architecture is that most of deep-pipelined kernels can be mapped into one RC rather than across several RCs. As a result, the communication overhead among RCs can be decreased so that the computing efficiency can be improved. Similar to MORE, there is no global memory for data caching in the RC array and therefore it suffers from the stream locality problem as the MORE architecture.

Different from the previous architectures, BiRC [7] is similar to commercial FPGA architectures by placing the same type of RCs in the same column. There are totally three types of RCs: arithmetic logic unit (ALU), memory block and multiplier. Each RC is fully connected with the neighbors in four directions by multiple transmission channels. The flexibility of interconnections inevitably increases the area overhead and power consumption. In addition, BiRC adopts a static mapping strategy while the dynamic scheduling is utilized with execution triggering. As a result, the trigger signals have to be generated by one or more RCs. This kind of control flow is at the cost of RC resource and may lead to low resource utilization.

Based on the above analysis, a new CGRA architecture is proposed in this paper with the following characteristics. Firstly, each RC in the proposed CGRA contains multiple PEs, such as ALUs, multipliers and memory blocks. This design strategy provided the ability to map kernel expressions onto one RC instead of different RCs. That is similar to [6], but the major difference is that all the PEs within each RC are organized as a "map-reduce" style in our architecture rather than the chain-style in [6]. Secondly, in addition to the local memory blocks in each RC, global stream register files (SRFs) are provided for data caching in the RC array. Thirdly, an elastic interconnection method proposed in [12] is used for RC interconnections to support RC combination and decomposition. Finally, the RC array is optimized for stream applications, especially for machine learning applications. To improve the execution efficiency of the proposed CGRA for stream applications, a new programming model called dual-track model is proposed. The model consists of two steps: the first step is the static configuration to determine RC functionalities; the second step is the dynamic configuration to manage the input data streams. The CGRA architecture based on this model is named as dual-track CGRA or DT-CGRA for short in the following sections.

III. PROGRAMMING MODEL

We derive the programming model from the intuition observation of the image processing domain. Many processing procedures can be regarded as kernel functions executing over a limited scope of the input data, and then shifting to the next scope in a specific order. Fig. 1(a) shows an example of a simple procedure, where the kernel scope is in a diamond shape. The complex procedures can be represented as the cascade of multiple simple procedures, as illustrated in Fig. 1(b). In machine learning algorithms, complex procedures are very common. Examples are the pyramid image processing in SIFT algorithms [23] and the multi-layers convolutions in CNN [13].

The simple procedure in Fig. 1(a) can be expressed by the pseudo code in Fig. 2, which consists of three loops. Two outermost loops, loop 1 and loop 2 represent the scope scanning behavior, i.e., shifting from left to right and from top to bottom with strides Sr and Sc. The innermost loop, loop 3 is the kernel process. The NUM is the number of input data needed to...
compute the kernel expression in the scope. From Fig. 2 we can see that the functionality of the kernel remains the same, while only the input data are different. Inspired by this observation, we propose the dual-track programming model in Fig. 3. There are two steps in the model. The first step is to construct the functionality of a given kernel by the static configurations. The second step is to manage the data stream by the dynamical configurations. The dynamic configuration contexts can be regarded as VLIW instructions. After configuration by the VLIW instructions, the mapped RCs can execute in the stream style until the end of data loading.

To demonstrate the proposed model, a convolutional operation example that is widely used in the image processing and machine learning domain is presented. Fig. 4 shows the static mapping and dynamic scheduling of the data stream for the convolution operation. There are two rows of data in the input image \( I_{map} \), denoted as \( L1 \) and \( L2 \) in Fig. 4. The size of the convolutional kernel is \( 1 \times 3 \). Based on the dual-track model, the convolution operation is statically mapped to one RC, e.g. RC1. Then the scheduling of the data stream in the convolution operation is compiled into 2 VLIW instructions, \textit{Instruction 1} and \textit{Instruction 2} in Fig. 4. For \textit{Instruction 1}, \( L1 \) of \( I_{map} \) is read out from memory and sent to RC1. At the same time, the output results are stored back to the memory. \textit{Instruction 2} performs the same operations except that the input data is \( L2 \). In this method, we can see that the convolution operation can be efficiently computed.

IV. ARCHITECTURE OF DT-CGRA

A. The overview of DT-CGRA

The proposed DT-CGRA is shown in Fig. 5. It mainly consists of two parts. There are several stream buffer units (SBUs) in the first part to cache the input/output data and intermediate results. The second part is the computing array (CA), which includes several columns of RCs and one column of special RCs, shown as IRCs and PRCs in Fig. 5. These special RCs are used to calculate interpolations and power functions such as \( x^{1/2} \), \( x^{-1/2} \) and \( x^{-1} \). The RCs are connected with each other by elastic interconnections, marked as blue arrows. These interconnections can support the RC decomposition and combination which are illustrated in Section IV-D. Besides, each row of RCs are connected through a multi-channel data bus marked as red arrows in Fig. 5. All the multi-channel data buses are connected with SBUs by a crossbar. There are totally four interfaces for the proposed CGRA. The external memory DMA interface provides a direct access to the off-chip memory for the SBUs. Another interface is provided for the scalability purpose through the network-on-chip communication protocol. With this interface, data in SBUs can be accessed by other CGRAs rather than across off-chip memory. In addition, there are two interfaces to fetch the correspondent dynamic and static VLIW instructions from the off-chip memory.

B. The RC architecture

According to the roofline model in [8], the provided memory bandwidth and the computation to communication ratio are two key factors that determine the architecture performance. In order to increase the computation to communication ratio, the interconnection of processing elements (PE) in each RC should provide data-reuse mechanisms. Besides, the interconnection of the PEs should provide the flexibility to support RC decomposition and combination to increase the resource utilization. On the other hand, inspired by [9], the idea of “map-reduce” can be applied to reduce the number of output interfaces and as a result, the bandwidths requirement can be decreased. Based on the above analysis, we empirically design the RC...
The functionalit of the controller in each RC is statically. Among them, PRC is designed to calculate the inverse square root, square root as well as the reciprocal based on the fast inverse square root algorithms in [10]. On the other hand, IRC is applied to calculate the interpolation for transcendental functions that can be approximated by piecewise functions. In Fig. 7(b), the input data $x$ is firstly compared with the boundary values of each interval from $x_0$ to $x_N$ in pipeline in order to generate an address for look up tables. The coefficients of $a_n$ and $b_n$, which are read out from look up tables are used to calculate the interpolation result, $a_n \times x + b_n$.

D. The interconnection of RCs

There are two major types of interconnections. One is the direct links among RCs, and the other is the crossbar between SBU s and the CA. The main strategy for these interconnections is the elastic data transmission mechanism, which can simplify the control procedure by converting the dynamic scheduling to the dataflow control [11]. Fig. 8 illustrates the elastic data transmission process proposed in [12]. The “Stop” and “Valid” signals control the handshaking process and ensure that the data from the sender can be safely accepted by the receiver. The status of the receiver is determined directly by the input dataflow without any extra data scheduling.

The elastic data transmission strategy provides the ability to merge two adjacent RCs to map more complicated kernels. Assuming that five multiplier-ALUs are required to implement an expression $E$, three multiplier-ALUs in the first RC and two more multiplier-ALUs in the next RC can be combined together.
to map $E$. The remains of the second RC can be further combined with the other RCs. Fig. 9 illustrates such an approach, where five RCs in two adjacent rows are configured to execute three such expressions in parallel, as shown in yellow, green and red regions respectively.

Comparing to the CA interconnections that are configured statically, the crossbar in Fig. 5 is controlled by the dynamic VLIW instructions after the configurations of SBUs. The major function of the crossbar is to assist the SBUs in issuing read or write operations to the CA, which is discussed in the next subsection.

E. The SBU architecture

In Fig. 5, every SBU can be regarded as a global stream data buffer that is transparent to all the RCs. Each SBU contains a stream register file (SRF) and an address generator to generate read and write addresses simultaneously. The generated addresses are then sent to the dual port SRF or the off-chip memory to load/store data streams. For simplicity, only two modes, namely sequential mode and fixed-step mode, are supported in this architecture. For the sequential mode, the data are read/written from/to the start address continuously until the end of the address. For the fixed-step mode, the addresses are generated from beginning to the end with a fixed step. An example is given in Fig. 10 with a read operation to the SRF.

All the behaviors of SBUs, such as read/write from/to the CA or the off-chip memory, are controlled by the dynamic VLIW instructions. To demonstrate the control flow of data streams, we 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. 11. In the first VLIW instruction cycle, the $k$-th SBU and the control signals correspondent 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 SRF for the $RC_{ij}$. At the same time, the next VLIW instruction is fetched and then the correspondent SBU is configured after the end of its address generation.

Normally, the clock cycle numbers to load or store data may vary from 10 to 1000 in stream applications, especially for image processing and machine learning domains. In addition, with the help of the double-buffer technique in the dynamic configuration process, the dynamic configuration overhead is very small which can be neglected.

F. Implementation details

As an exploration, the implemented CA includes $5 \times 4$ RCs and one column of special RCs. The design parameters of FIFOs, SRAMs, the numbers of SBUs, RCs and the special RCs are shown in TABLE I. The architecture is designed in Verilog and synthesized with Synopsys Design Compiler, which is then placed and routed by Synopsys IC Compiler based on the SIMC 55 nm HVT and LVT libraries. The power consumption is also estimated by IC Compiler. The static configuration contexts and dynamic configuration instructions are written in microcode format manually.

V. ALGORITHM MAPPING EXAMPLES

In machine learning domains, convolution operation and large-scale matrix multiplication are two of the most common
A. There are two phases for data scheduling. 

(b) The computation process of the i-th RC in #Phase 0. 

(c) The computation process of the i-th RC in #Phase 1. 

Fig. 12: A 5 × 5 convolutional kernel is mapped into three RCs which includes two phases. Each phase is correspondent to a dynamic configuration instruction. 

computing patterns. For example, the convolutional layers and the fully connected layers account for nearly 90% of the computation time according to [14] in the state-of-the-art convolutional neural network (CNN) [13]. In this section, several strategies are demonstrated to map computing kernels to the proposed DT-CGRA based on the convolution operations and matrix multiplications required in the convolutional layers and the fully connected layers. It should be mentioned that these strategies can be adapted to other algorithms which are not limited to the demonstrated examples. 

A. Mapping strategy of the convolution layers 

The strategy to map convolution operations in the convolutional layers can support flexible sizes of convolutional kernels with various strides. Without loss of generality, we suppose that the convolutional kernel is in the size of 5 × 5 and the stride is 2 × 2. The width of the 2D input map is N. Then the number of multiplier-ALUs that are required to map a convolutional kernel is determined by \( \left[ \frac{5}{2} \right] \times \left[ \frac{5}{2} \right] \). There are two phases for data scheduling, where each phase is correspondent to one dynamic VLIW configuration instruction due to the change of input data streams. Fig. 12 shows the mapping and computing strategy. The convolutional kernel is mapped into three RCs with two phases, as shown in Fig. 12 (a). Before the computation begins, the weights of the convolutional kernel are loaded into three SRAM blocks embedded in each RC. 

To simplify the control procedure, the convolutional kernel size is extended to be 6 × 6 by padding 0s at the right and bottom sides of the convolutional kernel. In detail, the weights in the first two rows of the extended convolutional kernel, \((w_{00}, w_{01}, \ldots, w_{0,6})\) and \((w_{10}, w_{11}, \ldots, w_{1,4}, 0)\), 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 kernel weights are loaded, the input data are read out from the source SBU in row order and broadcasted to all the mapped RCs. As for the even rows of the input map, three RCs work in #Phase 0. For the odd rows, the mapped RCs work in #Phase 1. The computation processes of these two phases are illustrated in Fig. 12 (b) and (c) respectively, where \( L^{m}_{i0} \) and \( L^{out}_{i0} \) represent the input and output of intermediate results cached in the i-th SBU at phase \( p \), \( i=0, 1, 2, p=0, 1 \). For example, the first row of the input data are loaded into the first RC to do convolution with the first row of the weights. The generated results are cached in the 0-th SBU. After the end of #Phase 0, the first RC works 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 cached results from #Phase 0 are read out from 0-th SBU to generate the final outputs of #Phase 1, which are then saved to the 0-th SBU again and used for the second RC in #Phase 0. During this process, we can see that the input data are fully reused and the convolutions are computed in parallel. 

Generally speaking, suppose the size of convolutional kernel is \( k_x \times k_y \) with the stride of \( s_x \times s_y \). \[
\left[ \frac{k_x}{s_x} \right] \times \left[ \frac{k_y}{s_y} \right]
\]
multiplier-ALUs are needed according to the above mapping strategy. The decomposition and combination characteristic of RCs introduced in Section IV-D provides the ability to map the convolutional kernels flexibly and efficiently. 

B. Mapping strategy of the fully connected layers 

The major computing pattern in the fully connective layer is the matrix multiplication. For example, the dimensions of input features and the output results are 9216 and 4096 respectively in the first fully connected layer of AlexNet [13], which can be expressed by a large scale matrix multiplication as 

\[
O_{1 \times 4096} = I_{1 \times 9216} \cdot W_{9216 \times 4096}
\]

If the input feature \( I \) from the previous layer is directly used to compute the output results \( O \), the system suffers from the frequently loading of the weight \( W \) from SBU to SRAM blocks in RCs, which inevitably introduces large overhead on data transmission and ultimately decreases the overall performance. One intuition idea is to increase the “lifecycle” of the weights stored in the RCs. A direct method is to adopt the “batch” strategy, which means a certain number of input features, e.g. 100, are batched together to construct a much larger matrix, such as \( I_{100 \times 9216} \). In this way, the lifecycle of the weights can be
improved 100 times so that the time cost to load new weights into the SRAM blocks can be hidden by the computation time. With the capacity limitation of each SRAM block, a matrix partition method is adopted to divide the weight matrix into several smaller blocks so that each block can be loaded into one SRAM. In order to adopt double-buffer strategy in SRAMs, the dimension of each sub-matrix is limited to 128. As a result, the input feature matrix $I_{100 \times 9216}$ is divided into 100 $\times$ 72 blocks. Each block is denoted as $I_{1 \times 128}$, where $i$ and $j$ are the block indexes. Considering the fact that the total number of SRAMs in all RCs is 60, the weight matrix can also be divided into two types. One is $W_{m \times n}^{1 \times 60}$, and the other is $W_{m \times n}^{2 \times 64}$, where $m$ and $n$ are the indexes, $m=0,1,\ldots,71, n=0,1,68$. With these two techniques, the matrix multiplication can be expressed as

$$O_{100 \times 4096} = \begin{pmatrix} I_{100}^0 & I_{100}^1 & \ldots & I_{100}^{99} \\ I_{100}^{100} & I_{100}^{101} & \ldots & I_{100}^{199} \\ \vdots & \vdots & \ddots & \vdots \\ I_{100}^{200} & I_{100}^{201} & \ldots & I_{100}^{299} \end{pmatrix} \begin{pmatrix} W_{1}^{0,0} & W_{1}^{0,1} & \ldots & W_{1}^{0,67} & W_{1}^{0,68} \\ W_{1}^{1,0} & W_{1}^{1,1} & \ldots & W_{1}^{1,67} & W_{1}^{1,68} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ W_{1}^{100,0} & W_{1}^{100,1} & \ldots & W_{1}^{100,67} & W_{1}^{100,68} \end{pmatrix} \begin{pmatrix} Y_{1}^{0} \\ Y_{1}^{1} \\ \vdots \\ Y_{1}^{100} \end{pmatrix} \tag{2}$$

To illustrate the computing process of equation (2) on the DT-CGRA, we take the multiplication process of $I_{100 \times 9216}$ with $W_{1}^{0,0}$ for example. Firstly, 60 columns of $W_{1}^{0,0}$ are loaded into 60 SRAMs in RC units. Then the sub-matrices $I_{100}^0, I_{100}^1, \ldots, I_{100}^{99}$ in the first column of the input feature matrix $I_{100 \times 9216}$ are streamed and broadcasted to all the RCs to calculate the intermediate products, which are used to add up with the multiplication results of the second sub-matrix $W_{1}^{1,0}$ by the second column of $I_{100 \times 9216}$, i.e. $I_{100}^{100}, I_{100}^{101}, \ldots, I_{100}^{199}$, this process is repeated until the final matrix result is completed. This computing process can be mapped into the DT-CGRA as shown in Fig. 13.

VI. EXPERIMENTAL RESULTS

The DT-CGRA is designed for the stream application domain, especially for machine learning algorithms. Generally speaking, object inference in machine learning algorithms contains three major stages, including feature extraction, feature selection and inference, as shown in Fig. 14. In this paper, several representative algorithms covering these three stages are selected to verify our architecture. Among them, histogram of oriented gradients (HOG) [15] and CNN algorithms are used for feature extraction; k-means, spatial pyramid model (SPM) [16] and principal component analysis (PCA) are adopted for feature selection while linear vector machine (SVM), Softmax [13] and Joint Bayesian [17] are used for inference. The implementation and the application scenarios of these algorithms are listed in Table II. All of these algorithms are evaluated in the test stage and the training stage is not considered in this paper.

For comparison purpose, the above algorithms are mapped on the proposed DT-CGRA, which are compared with the CPU solutions and some of the highly customized FPGA implementations. The CPU solutions are implemented on the Intel i7-3770 (3.40 GHz) processor with single thread based on CentOS 6.6 64-bit operating system.

A. DT-CGRA results

The proposed DT-CGRA architecture is implemented in Verilog and synthesized, placed & routed based on SMIC 55 nm process. The final results reported by IC Compiler show that the area of the proposed architecture is $3.791,206.12$ $\mu$m$^2$ with the estimated total dynamic power consumption of 1.79 $W$. The delay of the critical path is 1.95 $ns$ which means the architecture can run at 500 Mhz. The overall architecture contains 71 multipliers and 119adders and as a result, the peak performance can reach 95 GOP/s. The breakdown of the area and power consumptions on each component is listed in Table III, where memories and registers account for nearly 85% of the total power.
B. Comparison to CPU

The algorithms in Table II are evaluated both on the CPU and the proposed DT-CGRA architecture. The results are shown in Fig. 15, from which we can see that the proposed architecture outperforms the CPU solution in all selected algorithms with average speedup of 38.86x. Among them, the maximum speedup is achieved by PCCA algorithm. It is not surprising to see this result because our architecture is more capable to handle the data-locality and data-parallelism operations than the CPU. In the test stage, the key operation of the PCCA algorithm is the matrix multiplication. Based on the proposed “batch” strategy and the matrix partition method, the PCCA algorithm can be executed efficiently. The implementation that obtains the lowest speedup compared to CPU is the HOG algorithm. The main reasons are the follows. Firstly, the parallelism of dataflow are limited due to the special operations in the HOG algorithm, e.g., magnitude and orientation calculations. Secondly, it is inefficient to implement histograms on the RCs because of the sparsity nature of histogram operations. For example, it requires 9 RCs to vote for the 9-dimensional features. As a result, the proposed CA can only support histogram of two neighboring windows simultaneously.

The energy consumptions of the proposed architecture are also evaluated. Fig. 16 depicts the energy consumption of CPU over the energy consumption of DT-CGRA on eight algorithms. Results show that the proposed architecture can achieve average 1442.71x more energy efficient than the CPU solutions.

C. Comparison to highly customized implementation

The performance of the proposed architecture is further compared with several outstanding solutions for dedicated algorithms, as shown in Table IV. An ASIC approach called ShiDianNao [18], which targets at various CNN accelerations in camera sensor applications, is compared with our architecture in terms of speedup and power consumption. Reference [18] has reported that it takes 0.047 ms to process a 64×36 patch by the convNN algorithm with the power consumption of 320 mW. It outperforms DT-CGRA with 6.90x energy efficiency. This result is due to their fully optimizations on computation architecture and memory hierarchy for application specific algorithm. However, DT-CGRA is more flexible due to the capability to map other stream applications.

We also compare the implementation results of AlexNet on DT-CGRA with a high performance FPGA solution reported in [19]. The adopted FPGA of [19] is Virtex-7 XC7VX485T with a 28 nm process technology. As shown in Table IV, it takes 21.61 ms to compute five convolutional layers with the power of 18.61 W. However, the proposed DT-CGRA outperforms [19] with 7.82x energy efficiency.

| Table IV: Comparison with Highly Customized Solutions |
|-----------------|-----------------|-----------------|-----------------|-----------------|
| Solution on convNN | Time (ms) | Power (W) | Peak Performance (GOP/s) | Energy-efficiency |
| ShiDianNao [18] | 0.047 | 0.32 | 194 | 6.90x |
| DT-CGRA | 0.058 | 1.79 | 95 | 1x |
| Solution on Alex-Net | Time (ms) | Power (W) | Peak Performance (GOP/s) | Energy-efficiency |
| FPGA2015 [19] | 21.61 | 18.61 | 61.62 | 1x |
| DT-CGRA | 28.72 | 1.79 | 95 | 7.82x |

VII. CONCLUSIONS

In this paper, we propose a dual-track coarse-grained reconfigurable architecture which aims for the stream applications, especially for the machine learning domain. The proposed DT-CGRA implemented with the SMIC 55 nm process can run at 500 MHz with the estimated power of 1.79 W and the footprint of 3.79 mm². The DT-CGRA is evaluated by selecting eight machine learning algorithms, and the comparisons show that the DT-CGRA can achieve 1443x energy efficiency comparing to CPU solutions and 7.82x energy efficiency comparing to a high performance FPGA solution.

ACKNOWLEDGEMENTS

This work is supported by National Natural Science Foundation of China 61131001. We also appreciate the careful works and the constructive suggestions of the anonymous reviewers.

REFERENCES


