NP-CGRA: Extending CGRAs for Efficient Processing of Light-weight Deep Neural Networks

Jungi Lee and Jongeun Lee*
Dept. of Electrical Engineering, Ulsan National Institute of Science and Technology (UNIST), Ulsan, Korea
Neural Processing Research Center, Seoul National University, Seoul, Korea

***Abstract*—Coarse-grained reconfigurable architectures (CGRAs) can provide both high energy efficiency and flexibility, making them well-suited for machine learning applications. However previous work on CGRAs has a very limited support for deep neural networks (DNNs), especially for recent light-weight models such as depthwise separable convolution (DSC), which are an important workload for mobile environment. In this paper, we propose a set of architecture extensions and a mapping scheme to greatly enhance CGRA's performance for DSC kernels. Our experimental results using MobileNets demonstrate that our proposed CGRA enhancement can deliver 8∼18× improvement in area-delay product depending on layer type, over a baseline CGRA with a state-of-the-art CGRA compiler. Moreover, our proposed CGRA architecture can also speed up 3D convolution with similar efficiency as previous work, demonstrating the effectiveness of our architectural features beyond DSC layers.**

I. Introduction

Recently a number of DNN (Deep Neural Network) processing units, or DPUs, have been proposed, which can be classified as soft DPUs (implemented on an FPGA) and hard DPUs (fabricated into a chip). Hard DPUs (e.g., TPU [1]) can have higher performance and energy efficiency but lack flexibility, and may be difficult to support future application changes. Soft DPUs (e.g., [2]–[4]) can be easily upgraded, but typically have much lower performance per cost and energy efficiency compared to hard DPUs. CGRAs can strike a balance between energy efficiency and flexibility, such as supporting new activation functions (e.g., leaky ReLU [5]) and skip connections. Also CGRAs can be utilized for other applications than DNNs.

Previous work on mapping DNNs to CGRAs includes new architectures [6], [7] and a new compilation method [8], but they all target 3D convolution only (such as used in AlexNet [9]) [6]–[8]. However, for mobile applications, conventional 3D convolutions are superseded by light-weight models exploiting depthwise separable convolution (DSC) such as MobileNets [10], [11] due to their significantly higher inference performance and greatly reduced model size and computation complexity. Depthwise separable convolution is realized as a combination of depthwise convolution (DWC) and pointwise convolution (PWC) layers. While PWC typically accounts for over 90% MAC operations, in terms of runtime DWC can account for up to 40% due to its low computation-to-data-transfer ratio and difficulty in mapping DWC. Hence it is important to provide optimized mapping for DWC as well as PWC.

In this paper we first present our analysis showing that CGRAs are not necessarily slower than hard DPUs when it comes to machine learning workload, if a right set of architectural features are provided. Based on the analysis, we present three generic architecture extensions for CGRAs—crossbar-style memory bus, dual-mode MAC (multiply-accumulate) unit, and operand reuse network—along with a mapping scheme that can greatly enhance CGRA's performance for DSC kernels.

Our experimental results using MobileNet V1 and V2 [12], [13] demonstrate that our proposed features can improve the efficiency of CGRA for DWC and PWC layers by 8× and 18×, respectively, in terms of area-delay product (ADP) over a compiler approach [14]. Moreover, though not explicitly optimized for, 3D convolution on our architecture is also quite efficient, generating competitive performance and ADP as a CGRA [7] explicitly optimized for machine learning algorithms including 3D convolution.

In this paper we make the following contributions. First we analyze the performance bottleneck of CGRAs for DNN acceleration. Second we propose a small set of generic architecture extensions and a mapping scheme for DWC and PWC kernels. Third we evaluate our proposed CGRA called *NP-CGRA (Neural Processing-CGRA)* for MobileNet models.

II. Related Work

A. Baseline CGRA

While CGRA is a generic term encompassing many different architectures [6]–[8], [14]–[16], we consider ADRES-like CGRAs [16] as our baseline, which have been most extensively studied. The main datapath consists of a 2D array of PEs (Processing Elements) interconnected with a mesh-like network, plus local memory implemented as multi-banked SRAM blocks for high on-chip bandwidth. PEs can perform arithmetic/logic and memory operations though details vary. The PE operations and inter-PE connections are dynamically reconfigurable with no runtime overhead, thereby supporting pipelining of loops with II (Initiation Interval) greater than 1.

This work was supported by Samsung Advanced Institute of Technology, IITP grants funded by MSIT of Korea (No.2020-0-01336, Artificial Intelligence Graduate School Program (UNIST), and No.1711080972, Neuromorphic Computing Software Platform for Artificial Intelligence Systems), and Free Innovative Research Fund of UNIST (1.170067.01).
*J. Lee is the corresponding author of this paper (Email: jlee@unist.ac.kr).

978-3-9819263-5-4/DATE21/©2021 EDAA

TABLE I: Theoretical min latency (ms, sum of 7 DWC layers)

Architecture	Compute time	L1 transfer	Layer latency
CGRA baseline (4x4)	1.68	0.75∼4.10	1.68∼4.10
CGRA enhanced (8x8)	0.21	0.19	0.21
Eyeriss (168 PEs)	0.20	0.23	0.23

There are two kinds of memory operations on CGRAs in the literature: addressed vs streamed load-store. Addressed load-store [16] is more common among CGRA compilers as it supports random memory access, but requires explicit address computation (which uses PE cycles). Streamed load-store [15] requires dedicated AGUs (Address Generation Units), which support only a limited set of access patterns. In either case, it is possible for all connected PEs to simultaneously read a memory bus if needed.

B. CGRA Architecture Exploration

CGRA architecture exploration has been performed in [17], which however does not take into account DNN workload or specific mapping schemes. While single-cycle MAC operation is common in DPUs, it is rarely supported on CGRAs by default. Our dual-mode MAC is configurable at the application granularity to minimize cycle time impact of operation chaining. An extreme version of operation chaining has been proposed [18] in order to accelerate narrow acyclic subgraphs at subcycle granularity, which however complicates datapath, control, and compiler scheduling significantly. Our operand reuse network is an input-to-input network whereas operand networks in the literature [19], [20] generally refer to output-to-input networks.

C. DPU Optimization

DSC computation has been targeted by both hard DPUs [13] and soft DPUs, but not by CGRAs. We do not consider pruning [21] directly in this work, since DSC is already a form of sparsity at coarse granularity [22] while being much more amenable to hardware parallelization than fine-grained sparsity. We also do not consider aggressive quantization, but the width of datapath is trivially configurable at design time.

III. NP-CGRA Architecture

A. CGRA Performance Bottleneck Analysis

Table I compares a baseline CGRA [8] with Eyeriss [12], a reference hard DPU, in terms of minimum theoretical latency, using 7 DWC layers from MobileNet V2, one from each bottleneck (we see similar results with other layers as well). The baseline CGRA has 4x4 PEs and runs at 500 MHz with 4-byte word size, and Eyeriss has 168 PEs and runs at 200 MHz with 2-byte word size.

We calculate minimum theoretical latency simply by the max of compute time (assuming 100% PE utilization), L1 transfer time (i.e., on-chip memory access latency), and external memory DMA (Direct Memory Access) time, the last of which is very small for all the cases compared, and not shown. To estimate L1 transfer time for the baseline CGRA, we assume all 4 load-store units (one per row) are 100% utilized, and consider two scenarios: the least and most data reuse of IFM (Input Feature Map). For Eyeriss we assume 32 load-store units, and most data reuse.

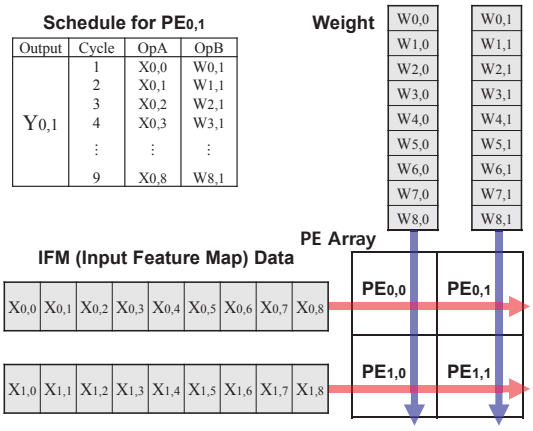


Fig. 1: Mapping PWC (or matrix mult.) to a 2×2 CGRA.

Our result suggests that there is ∼8× compute time difference between the baseline CGRA and Eyeriss DPU even if we assume 100% PE utilization, which may be harder for CGRA. The difference grows if CGRA fails to reuse IFM data optimally.

To fill the gap, we consider *CGRA enhanced*, which is the same CGRA but with 8x8 PEs and 2-byte word size. Also the PEs of CGRA enhanced can do MAC operation in a single cycle like Eyeriss (CGRA baseline can do either MUL or ADD, not both). These changes can bring compute time to Eyeriss-level, but layer performance would still suffer due to L1 transfer bottleneck. To make it compute-bound, CGRA enhanced needs to have 16 load-store units, *one per row or column*, and the most data reuse scenario.

To summarize, our analysis suggests that CGRA is capable of delivering hard DPU-level performance, but needs a few major changes: single-cycle MAC, larger array size, at least 2× on-chip memory bandwidth, and extremely high PE utilization.

B. Our Proposed Architecture Extension

Our driving application is pointwise convolution (PWC), which is also known as 1x1 convolution and algorithmically equivalent to matrix multiplication. While one can use a CGRA compiler (e.g., [23], [24]) to compile matrix multiplication for a CGRA, it would yield a vastly suboptimal schedule. In case of matrix multiplication, it is straightforward to find an optimal schedule manually, if one is allowed to modify the architecture slightly. The most critical architectural change is crossbar-type memory busses, as opposed to parallel busses.

1) Crossbar-style Memory Bus: Figure 1 illustrates our proposed mapping for a 2×2 CGRA, in which we use the first 2 rows of one source matrix (X) and the first 2 columns of the other source matrix (W), to generate the top-left 2×2 submatrix of the result matrix. The result submatrix is

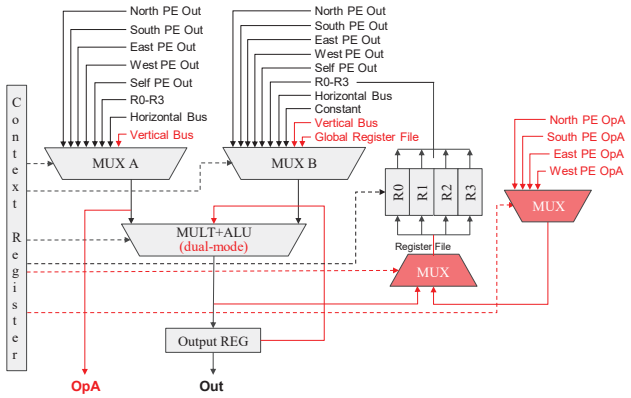


Fig. 2: Proposed PE architecture (our extension shown in red).

generated on the 2×2 PEs through a series of MAC operations (thus output stationary), as indicated by the schedule.

To provide the four PEs with correct operands, all we need is two horizontal busses and two vertical busses. Note that the data on a bus can be accessed by all connected PEs, and we add only vertical busses; horizontal busses already exist. For instance, $\text{PE}_{0,0}$ and $\text{PE}_{0,1}$ can access the same $\text{X}(0, i)$ at cycle i ($0 \leq i \leq 8$) through a horizontal bus (called H-bus), and similarly, $\text{PE}_{0,0}$ and $\text{PE}_{1,0}$ share $\text{W}(i, 0)$ through a vertical bus (V-bus). To use all PEs for MAC operations, streamed load-store is necessary. This mapping achieves 100% PE utilization, each PE performing MUL (multiplication) *and* ADD (addition) operations every cycle, given dual-mode MAC units, explained next.

2) Dual-mode MAC: In most CGRAs a PE performs only one operation per cycle, either MUL or ADD, which is fine if they are used intermittently. We propose configurable chaining of MUL and ADD operations, which can reduce PWC latency to half, though it may also increase cycle time. We make chaining configurable at the application granularity, so that higher clock speed is selected if the application does not use MAC chaining. We call this *dual-mode MAC*. A detailed diagram of dual-mode MAC is omitted due to page limit, but it is straightforward to design one.

3) Operand Reuse Network: To make it easy to realize spatial data reuse on CGRAs we propose *operand reuse network*, which enables input-to-input routing as opposed to output-to-input routing. Consider an FIR filter example: $y_i \leftarrow w_0 x_i + w_1 x_{i+1} + w_2 x_{i+2}$, where i is the index variable of a loop that is pipelined. One way to map this loop to a CGRA is to place output variables y_i to different PEs (i.e., y_i to PE_i), called *output stationary*, and route input and coefficients to PEs. In this scheme the same input data is used by multiple PEs at different cycles (e.g., x_2 is used by PE_0, PE_1, and PE_2 at consecutive cycles). Thus operand reuse network allows one of the source operands of a PE (i.e., the output of an input MUX) to be passed to neighbor PEs without affecting other computation that PEs may be doing, as illustrated in Figure 2.

While a weight stationary scheme could realize spatial data reuse without operand reuse network, it cannot easily utilize more PEs than the number of weight parameters. Also, the output stationary scheme is more amenable to 2D extension.

4) Other Changes: The crossbar-style memory bus implies that the local memory should be divided into two, V-MEM connected to V-bus and H-MEM connected to H-bus. We set the combined size of V-MEM and H-MEM equal to that of the baseline CGRA's local memory. Also AGUs are needed for streamed load-store. In addition, for efficient mapping of DWC with stride of 1, our architecture includes a small single-port global register file (GRF), which is used to broadcast DWC weights to all PEs. The index for the GRF is given in the configuration. GRF can be filled either by DMA or through a dedicated buffer, called Weight Buffer, which can be very small as it is used for DWC only.

TABLE II: Parameters

Symbol	Meaning
N_r, N_c	Number of rows/columns of a CGRA's PE array
K, S	Kernel size and stride of convolution
N_i, N_o	The number of input/output channels
N_h, N_w	The height and width of OFM (Output Feature Map)
$B_r \times B_c$	Number of tiles in the current block (row×column)

IV. Application Mapping for NP-CGRA: DWC Case

We now present our application mapping for DWC kernels (PWC mapping is already outlined in Section III-B1). Table II lists parameters. First we present a general method that works for any stride, then an optimized version for $S = 1$, which is most common. In the following, a *tile* refers to the amount of work (or corresponding data) that is done *simultaneously* by a CGRA, with its size determined by the CGRA size. *Block* is the amount of work that can be done using *local* data only. Block size is a multiple of tile size.

While in this paper we mainly describe PE scheduling and data routing, which is crucial for maximizing PE utilization and minimizing memory access, our implementation and evaluation results include complete mapping including data layout and AGU algorithms.

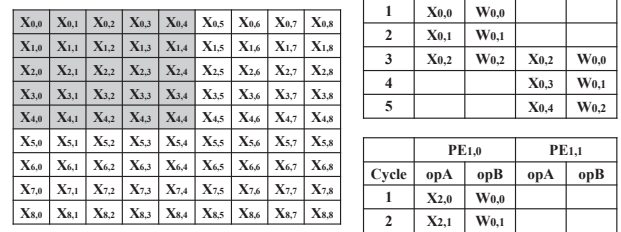
A. Depthwise Convolution with Arbitrary Stride

Consider mapping DWC with $K = 3$ to a 2×2 CGRA. Here we parallelize the computation of one channel across the PE array. The following equations reveal the terms needed to compute the first 2×2 output, an *output tile*.

$$y_{0,0} = w_{0,0}x_{0,0} + w_{0,1}x_{0,1} + w_{0,2}x_{0,2} + w_{1,0}x_{1,0} + \cdots + w_{2,2}x_{2,2}$$
$$y_{0,1} = w_{0,0}x_{0,2} + w_{0,1}x_{0,3} + w_{0,2}x_{0,4} + w_{1,0}x_{1,2} + \cdots + w_{2,2}x_{2,4}$$
$$y_{1,0} = w_{0,0}x_{2,0} + w_{0,1}x_{2,1} + w_{0,2}x_{2,2} + w_{1,0}x_{3,0} + \cdots + w_{2,2}x_{4,2}$$
$$y_{1,1} = w_{0,0}x_{2,2} + w_{0,1}x_{2,3} + w_{0,2}x_{2,4} + w_{1,0}x_{3,2} + \cdots + w_{2,2}x_{4,4}$$

The *input tile*, which is the set of IFM data needed to produce an output tile, is the gray-filled rectangle in Figure 3a.

Figure 3b illustrates our proposed schedule. For instance, to compute the top row of the output tile, the top three rows of the input tile are needed, which are given sequentially through an H-bus. Each PE performs MAC operations when they see the corresponding input data on the H-bus, which simplifies



$X_{0,0}$	$X_{0,1}$	$X_{0,2}$	$X_{0,3}$	$X_{0,4}$	$X_{0,5}$	$X_{0,6}$	$X_{0,7}$	$X_{0,8}$
$X_{1,0}$	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$	$X_{1,5}$	$X_{1,6}$	$X_{1,7}$	$X_{1,8}$
$X_{2,0}$	$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$	$X_{2,5}$	$X_{2,6}$	$X_{2,7}$	$X_{2,8}$
$X_{3,0}$	$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$	$X_{3,5}$	$X_{3,6}$	$X_{3,7}$	$X_{3,8}$
$X_{4,0}$	$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$	$X_{4,5}$	$X_{4,6}$	$X_{4,7}$	$X_{4,8}$
$X_{5,0}$	$X_{5,1}$	$X_{5,2}$	$X_{5,3}$	$X_{5,4}$	$X_{5,5}$	$X_{5,6}$	$X_{5,7}$	$X_{5,8}$
$X_{6,0}$	$X_{6,1}$	$X_{6,2}$	$X_{6,3}$	$X_{6,4}$	$X_{6,5}$	$X_{6,6}$	$X_{6,7}$	$X_{6,8}$
$X_{7,0}$	$X_{7,1}$	$X_{7,2}$	$X_{7,3}$	$X_{7,4}$	$X_{7,5}$	$X_{7,6}$	$X_{7,7}$	$X_{7,8}$
$X_{8,0}$	$X_{8,1}$	$X_{8,2}$	$X_{8,3}$	$X_{8,4}$	$X_{8,5}$	$X_{8,6}$	$X_{8,7}$	$X_{8,8}$

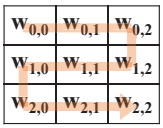
(a) IFM data (shown in gray is an input tile)

	PE0,0		PE0,1	
Cycle	opA	opB	opA	opB
1	$X_{0,0}$	$W_{0,0}$		
2	$X_{0,1}$	$W_{0,1}$		
3	$X_{0,2}$	$W_{0,2}$	$X_{0,2}$	$W_{0,0}$
4			$X_{0,3}$	$W_{0,1}$
5			$X_{0,4}$	$W_{0,2}$

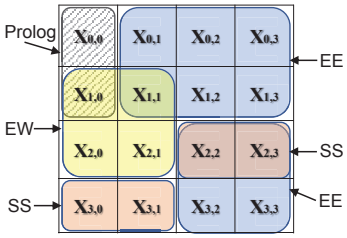
	PE1,0		PE1,1	
Cycle	opA	opB	opA	opB
1	$X_{2,0}$	$W_{0,0}$		
2	$X_{2,1}$	$W_{0,1}$		
3	$X_{2,2}$	$W_{0,2}$	$X_{2,2}$	$W_{0,0}$
4			$X_{2,3}$	$W_{0,1}$
5			$X_{2,4}$	$W_{0,2}$

(b) Schedule

Fig. 3: Mapping DWC on a 2×2 CGRA ($K = 3, S = 2$).



(a) Weight processing order



(c) How IFM data is accessed

	Weights	Phase	#cycles
1		Prolog	$N_c - 1$
2	$W_{00} \sim W_{02}$	EE	K
3	W_{12}	SS	1
4	$W_{11} \sim W_{10}$	EW	$K - 1$
5	W_{20}	SS	1
6	$W_{21} \sim W_{22}$	EE	$K - 1$

(b) Phase definition

Phase	Reuse dir.	Data loading (to which PEs)
Prolog		H-bus (all but first col)
EE	←	H-bus (Last col)
SS	↑	V-bus (Last row)
EW	→	H-bus (First col)

(d) Data reuse and loading

Fig. 4: Schedule and data movement for DWC ($S = 1$).

schedule. One can see that our schedule achieves maximal data reuse within each row, since the data needed for each row is presented only once. Weight parameters can be provided through V-busses because each column of PEs use the same weight parameters every cycle.

B. Depthwise Convolution with $S = 1$

Consider an example where $K = 3$ and the CGRA size is 2×2. Again we handle one channel at a time. Similar to the general version, our scheme is output stationary such that after a certain number of cycles the 2×2 PE array will contain the data for the first 2×2 output. The key problem is how to feed all the PEs with necessary input/weight data every cycle without oversubscribing memory access resources.

Figure 4 illustrates our solution. During the initial $N_c - 1$ cycles (called *prologue*), IFM data (the top-left $N_r \times (N_c - 1)$ submatrix) is loaded through H-busses into all PEs except the first column. For the next K cycles, the PE array processes the first row of the weight matrix using IFM data partially reused from the previous cycle (from the east-side PEs) and partially loaded from local memory (for the easternmost column), which is called EE (Expand East) phase. In the next cycle, the PE array processes $W_{1,2}$, which requires reusing IFM data from the south-side PEs and the southernmost PEs to load new IFM data, called SS (Shift South) phase. In the next $K - 1$ cycles, the PE array processes the remaining elements of the 2nd row of weight, which is similar to the EE phase except that we expand west, thus called EW (Expand West). This pattern of EE-SS-EW-SS is repeated until we finish processing all weight. In this schedule all PEs use the same weight element, which is provided by GRF, indexed by the CGRA controller.

This schedule takes $N_c - 1 + K^2$ cycles including prologue, except for initial memory streaming delay and final cycles for writing output data back to local memory. The data layout and AGU logic to support the above access pattern are a little complicated due to the SS phase. An alternative would be to load data for the southernmost PEs through H-bus over N_c cycles, which increases latency significantly. We place the full IFM data in H-MEM and the part needed for the SS phases in V-MEM. Loading data to both H-MEM and V-MEM is done by DMA.

TABLE III: NP-CGRA specifications

Number of PEs	64 (8×8)
Word size	16-bit
Clock frequency	500 MHz
Off-chip memory bandwidth	12.5 GB/s
DMA latency	200 cycles
H-MEM size (= V-MEM size)	39 KB (×2 sets)
Configuration memory size	9248 bytes (2312×32 bits)
Weight buffer size	1152 bytes (144×64 bits)

V. Experiments

A. Experimental Setup

To evaluate the effectiveness of our proposed architecture, we use MobileNets and compare against previous CGRA approaches as well as other DPUs. However, since MobileNet results are not reported by previous CGRA architectures, we also map AlexNet convolution layers to NP-CGRA and compare our result with those of previous CGRAs and DPUs as reported in the literature.

Our main comparison metric is inference throughput (frames/s) and cost efficiency (in ADP). We have developed a cycle-accurate simulator and also designed RTL for the baseline CGRA and our NP-CGRA, including PE array, AGUs, GRF, and the CGRA controller, which we have validated in terms of functionality and cycle-level behavior. For area estimation we have synthesized RTL designs with Synopsys Design Compiler using Samsung 65 nm standard-cell library. The on-chip memory area is estimated using Cacti 7.0 [25].

Table III lists specifications of NP-CGRA. The off-chip memory bandwidth is set to 12.5 GB/s as in SDT-CGRA [7]. H-MEM and V-MEM have the same size, which is set to $N_i K^2 \times N_r$ words, to make mapping AlexNet easier, although smaller memory sizes can also be accommodated by our mapping strategy. The number of configuration bits per cycle is 2312 = 36×64 + 8; each PE needs 4 more bits than a baseline PE due to increased input MUX sizes (1 bit) and the operand reuse network's MUXes (3 bits), and 8 more bits

globally for GRF index and to control streamed load-store. Weight Buffer, which is optional, is set to hold 64 copies of GRF contents.

B. Depthwise Separable Convolution Results

We use the first three layers right after the first standard convolution (i.e., 3D convolution) layer in MobileNet V1 [10] (width multiplier 1, resolution 224). We compare three cases:

- **Baseline+CCF**: Baseline CGRA with CCF compiler [23]
- **Matmul DWC**: NP-CGRA + Matrix multiplication-based DWC
- **Our mapping**: NP-CGRA + Our mapping scheme for PWC/DWC

For this experiment only, the CGRA size is set to 4×4 due to CCF compilation flow (for all three cases). The clock speed is 500 MHz for both the baseline and NP-CGRA.

The first case represents the state-of-the-art CGRA solution. For CCF, we apply loop pipelining to the loop level with the largest trip count, which is image height (N_h). The second case uses our mapping scheme for PWC only. DWC is converted into matrix multiplication by im2col, essentially using only one column of a CGRA, to which the K^2 dimension is mapped. The im2col time isn't taken into the account in this part.

Table IV summarizes the result. The architectural factor is about 2×, since our NP-CGRA has 2× faster arithmetic and memory operation rate than the baseline CGRA. So the large performance difference is attributed to mapping. A close look at the generated code has revealed that CCF generates extra 1 MUL and 3 ADD ops for every MAC operation (1 MUL, 1 ADD) in the program, which is due to address generation as it uses addressed load-store. Also the scheduled code has some empty slots, which further lowers the PE utilization. Overall, the mapping efficiency difference is about 10× in the case of PWC for the relatively small CGRA size. We expect the difference to increase for larger CGRA sizes. All in all, our NP-CGRA generates over 20× speed up and close to 18× ADP reduction for PWC over the baseline (our architecture has 18% larger total area including SRAM memory; synthesis result is discussed in Section V-C).

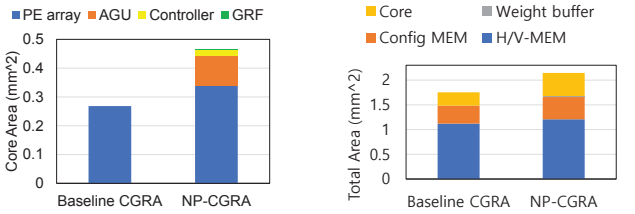
For DWC our NP-CGRA continues to deliver better performance and ADP than the baseline. While the utilization of the Matmul DWC case is around 16% (and cannot exceed 25% using only one CGRA column), our DWC mapping generates about 1.75∼3× higher performance and efficiency than the matmul-based mapping. Note that DWC (S=2) layers are the rarest in MobileNets while PWC accounts for the majority of MAC operations, which may justify relatively low effort optimizing for the former case.

C. Hardware Overhead Evaluation

Figure 5 compares the synthesis area of two 8×8 CGRAs at the target frequency of 500 MHz (timing met in both). The largest core increase comes from AGUs, which may be justified given the so many freed PEs by AGUs. The common logic and variables used by AGUs such as iterators

TABLE IV: MobileNet DSC result

Metric	Layer	CCF	Matmul DWC	Our mapping
Latency	PWC	78.91 (8.14)	3.72 (86.42)	3.72 (86.42)
(util)	DWC (S=1)	11.10 (8.14)	2.82 (16.04)	0.92 (49.00)
(ms,%)	DWC (S=2)	7.74 (5.83)	1.41 (16.01)	0.81 (28.00)
ADP	PWC	122.48	6.83	6.83
(mm^2·ms)	DWC (S=1)	17.22	5.17	1.69
	DWC (S=2)	12.02	2.59	1.48



(a) Core area comparison (b) Total area comparison

Fig. 5: Area comparison.

are implemented in the controller, shown in the graph. The increase in the PE array is modest (the baseline architecture has homogeneous operation set, meaning all PEs support MUL and ADD operations). Not surprisingly, the total area is dominated by SRAM memory, which puts the overall area overhead of NP-CGRA at 22.2%.

While we use the same clock frequency for both CGRAs in our ADP evaluation, our dual-mode MAC does increase the critical path delay. When driven for maximum speed, the critical path delay is increased from 1.23 ns (baseline) to 1.65 ns (NP-CGRA), which is due to the difference between MAC delay (1.08 ns) and MUL delay (0.68 ns). Considering the potential 2× increase in computation throughput, the 34% increase in cycle time seems justifiable. On the other hand, MAC operations are not utilized by current CGRA compiler (e.g., CCF), which can limit applicability.

D. Comparison with Previous Work Using MobileNet

No previous CGRA reports MobileNet or DSC performance. A few MobileNet accelerators for FPGAs exist but no reported ASIC area makes direct comparison difficult. Eyeriss v2 [13] targets MobileNet V1 with width multiplier 0.5 and resolution 128, which we compare in Table V. Eyeriss v2 has much more capable PEs than NP-CGRA, performing 2 MAC ops per cycle, which partially explains higher absolute performance compared with NP-CGRA. On the other hand, NP-CGRA is much smaller. While Eyeriss v2 reports gate count only, it appears larger than Eyeriss, so we assume Eyeriss v2 has the same area as Eyeriss. Also Eyeriss v2 uses 8-bit data width, we convert the area number to 16-bit equivalent by multiplying 2, which we believe is conservative. Overall, the NP-CGRA turns out to have lower ADP (2.22×) though it is due to its faster clock speed.

E. AlexNet Convolution Layer Results

While 3D convolution is not explicitly optimized for by our architecture, we map AlexNet convolution layers to NP-

TABLE V: Comparison with previous CGRA and DPU implementations

	Eyeriss [12]	Eyeriss-v2 [13]	Auto-tuning [8]	SDT-CGRA [7]	NP-CGRA (Ours)
Technology	ASIC (65 nm)	ASIC (65 nm)	CGRA (32 nm)	CGRA (55 nm)	CGRA (65 nm)
Clock frequency (MHz)	200	200	500	450	500
#PEs (#Ops/cycle)	168 (336)	192 (768)	16 (16)	25 (205)	64 (128)
Data width (bits)	16	8	32	16	16
On-chip data memory (kB)	108	192	320	54.6	156
Reported area (mm^2)	12.25	$\geq$ 12.25	1.55†	5.19	2.14
Converted area (65 nm, 16-bit) (mm^2)	12.25	$\geq$ 24.50	1.55†	7.25	2.14
MobileNet V1 (DSC runtime, ms)	-	0.78	-	-	4.01
MobileNet V2 (DSC runtime, ms)	-	-	-	-	18.06
MobileNet V1 ADP (DSC only, mm^2·ms)	-	19.11	-	-	8.60
AlexNet (Conv. runtime, ms)	28.82	9.79	990	23.24	40.07‡
AlexNet ADP (Conv. only, mm^2·ms)	353.03	239.96	1536.68	168.59	87.28‡

†Not reported in the paper, and assumed to be the area of the 4×4 baseline CGRA.
‡The ARM processor's runtime is included in latency but its area is *not* included in ADP.

CGRA, for quantitative comparisons with previous CGRA results as well as to see broader applications of our extensions outside DSC layers (see Table V). For NP-CGRA, we convert convolution into matrix multiplication using im2col and use PWC mapping. The im2col part is assumed to be done on the ARMv8 processor on Xilinx Ultra96-V2 board, which we have used to measure the runtime of im2col functions. The auto-tuning approach [8] applies various combinations of loop transformations (e.g., interchange, unrolling) to find the best loop nest for CGRA mapping, which is done by an in-house CGRA compiler. SDT-CGRA [7] is a novel architecture optimized for machine learning algorithms including convolutional neural networks (CNNs). Eyeriss [12] and Eyeriss v2 [13] are hard DPUs optimized for CNNs.

To allow comparisons among different technologies and data widths, we convert the reported areas into 65 nm, 16 bit-equivalents, which are then multiplied with runtime to calculate ADP. As expected, the auto-tuning approach has the lowest performance and efficiency, attributed to poor scheduling. Eyeriss and Eyeriss v2 are among the fastest while SDT-CGRA is the most efficient in terms of ADP, which is again due to its faster clock speed. Our NP-CGRA result does not include the area of the ARM processor, but it is quite competitive with other CGRA or DPU architectures in terms of both speed and ADP, demonstrating the efficacy of our extensions beyond DSC layers.

VI. Conclusion

We presented a set of generic architecture extensions for CGRAs that can greatly improve performance and efficiency for light-weight DNN models. We have also demonstrated that our proposed features are useful beyond DSC layers, such as for 3D convolution. We plan to apply our NP-CGRA to accelerating other machine learning algorithms and digital filters, many of which are based on matrix multiplication and convolution. Automatic generation of efficient code that exploits our proposed architectural features is left for future work.

References

[1] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th ISCA*, 2017, pp. 1–12.

[2] J. Fowers *et al.*, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th ISCA*. IEEE, 2018, pp. 1–14.

[3] A. Rahman *et al.*, "Efficient FPGA acceleration of convolutional neural networks using logical-3d compute array," in *DATE*, Mar. 2016, pp. 1393–1398.

[4] ——, "Design space exploration of FPGA accelerators for convolutional neural networks," in *DATE*, Mar. 2017, pp. 1147–1152.

[5] A. L. Maas *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *ICML*, 2013.

[6] M. Tanomoto *et al.*, "A CGRA-based approach for accelerating convolutional neural networks," in *2015 IEEE 9th MCSoc*, 2015.

[7] X. Fan *et al.*, "Stream processing dual-track CGRA for object inference," *IEEE Trans. VLSI*, vol. 26, no. 6, pp. 1098–1111, 2018.

[8] I. Bae *et al.*, "Auto-tuning CNNs for coarse-grained reconfigurable array-based accelerators," *IEEE TCAD*, vol. 37, no. 11, 2018.

[9] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Advances in NIPS*, 2012, pp. 1097–1105.

[10] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[11] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018, pp. 4510–4520.

[12] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[13] ——, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in CAS*, vol. 9, no. 2, pp. 292–308, 2019.

[14] S. Dave *et al.*, "RAMP: resource-aware mapping for CGRAs," in *DAC*. IEEE, 2018, pp. 1–6.

[15] H. Singh *et al.*, "Morphosys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE transactions on computers*, vol. 49, no. 5, pp. 465–481, 2000.

[16] B. Mei *et al.*, "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," in *International Conference on FPL*, 2003, pp. 61–70.

[17] D. Suh *et al.*, "Design space exploration and implementation of a high performance and low area coarse grained reconfigurable processor," in *2012 International Conference on FPT*. IEEE, 2012, pp. 67–70.

[18] Y. Park *et al.*, "CGRA express: accelerating execution using dynamic operation fusion," in *CASES*, 2009, pp. 271–280.

[19] M. B. Taylor *et al.*, "Scalar operand networks: On-chip interconnect for ilp in partitioned architectures," in *HPCA*. IEEE, 2003.

[20] J. Balfour *et al.*, "Operand registers and explicit operand forwarding," *IEEE Computer Architecture Letters*, vol. 8, no. 2, pp. 60–63, 2009.

[21] S. Han *et al.*, "Learning both weights and connections for efficient neural network," in *Advances in NIPS*, 2015, pp. 1135–1143.

[22] R. Zhao *et al.*, "Building efficient deep neural networks with unitary group convolutions," in *CVPR*, 2019.

[23] S. Dave *et al.*, "CCF: A CGRA compilation framework."

[24] S. A. Chin *et al.*, "CGRA-ME: A unified framework for CGRA modelling and exploration," in *ASAP*, 2017, pp. 184–189.

[25] R. Balasubramonian *et al.*, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM TACO*, vol. 14, no. 2, pp. 1–25, 2017.