Towards High-throughput Neural Network Inference with Computational BRAM on Nonvolatile FPGAs

Hao Zhang[a], Mengying Zhao[a*], Huichuan Zheng[a], Yuqing Xiong[a], Yuhao Zhang[b], Zhaoyan Shen[a]
[a] School of Computer Science and Technology, Shandong University, China
[b] Department of Computer Science and Technology, Tsinghua University, China

***Abstract*—Field-programmable gate arrays (FPGAs) have been widely used in artificial intelligence applications. As the capacity requirements of both computation and memory resources continuously increase, emerging nonvolatile memory has been proposed to replace static random access memory (SRAM) in FPGAs to build nonvolatile FPGAs (NV-FPGAs), which have advantages of high density and near-zero leakage power. Features of emerging nonvolatile memory should be fully explored to improve performance, energy efficiency as well as lifetime of NV-FPGAs. In this paper, we study an intrinsic characteristic of emerging nonvolatile memory, i.e., computing-in-memory, in nonvolatile block random access memory (BRAM) of NV-FPGAs. Specifically, we present a computational BRAM architecture (C-BRAM), and propose a computational density aware operator allocation strategy to fully utilize C-BRAM. Neural network inference is taken as an example to evaluate the proposed architecture and strategy, showing 68% and 62% improvement in computational density compared to traditional SRAM-based FPGA and existing NV-FPGA, respectively.**

***Index Terms*—FPGA, Computational BRAM, Neural network inference**

I. Introduction

Nowadays, due to advantages in terms of flexibility and energy efficiency, field-programmable gate arrays (FPGAs) attract high attention in areas of artificial intelligence (AI) and big data. A typical FPGA platform consists of configurable logic blocks (CLBs), embedded block random access memories (BRAMs), digital signal processing blocks (DSPs), and programmable routing resources. The intrinsic reconfigurability enables FPGAs to easily adapt to the evolving demands of diverse applications, leading to cost-effective and highly flexible solutions. With the fast development of AI models, the demands for both computation and memory resources in FPGAs are continuously increasing. On one hand, the rapid evolution of chip process technology has facilitated the transition of FPGA chips from the micrometre era to the nanometer era. On the other hand, more computation and memory resources are integrated into FPGA chips to adapt to large-data applications, as shown in Figure 1. With these trends, traditional static random access memory (SRAM) based FPGAs face significant challenges since SRAM has low density and high leakage power.

To solve this problem, it is proposed to replace SRAM in FPGAs with emerging nonvolatile memories (NVMs) to build nonvolatile FPGAs (NV-FPGAs) [2]–[6]. This leads to studies in circuit-, architecture-, as well as system-level design and optimization for NV-FPGAs, where features of NVMs are explored to optimize performance, energy efficiency, and lifetime of NV-FPGAs. For example, since representative NVMs have the lifetime issue, i.e., they can only afford a fixed number of writes, researchers have designed wear-leveling strategies to balance writes in NV-FPGAs to improve lifetime [2]. Considering the different working modes of NVMs, i.e., single-level cell (SLC) and multiple-level cell (MLC), researchers have explored the transformation of working modes in NV-FPGAs to achieve both performance and lifetime improvement [4], [6].

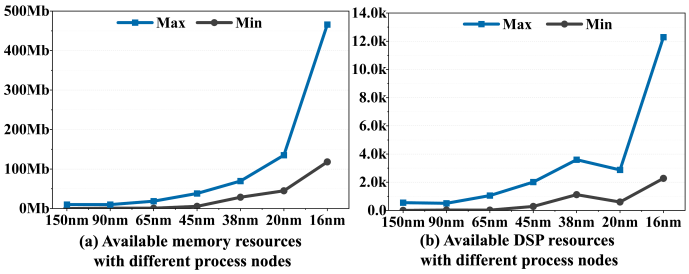

Fig. 1. More computation and memory resources are integrated into FPGA chips. Information is sourced from Xilinx Virtex Family product manuals [1].

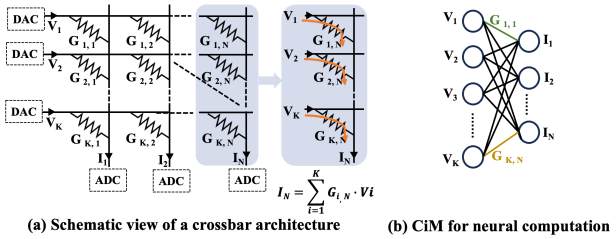

Fig. 2. Typical NVM-based CiM Architecture.

In this paper, we explore another feature of NVMs, i.e., computation-in-memory (CiM), to study the architecture design and synthesis flow optimization of NV-FPGAs. CiM is a new paradigm in which computation tasks are performed directly within the memory units, rather than transferring data between memory and processing units for computation [7], [8]. As shown in Figure 2, by mapping weight to the conductance of NVM cells and vectors on the input voltage, the NVM crossbar can perform matrix-vector-multiplication (MVM) op-

This work is supported by Natural Science Foundation of China (NSFC) (Grant No. 62372270). Mengying Zhao is the corresponding author. (e-mail: zhaomengying@sdu.edu.cn)

erations with high parallelism. This enables NVM to work in computation mode in addition to traditional storage mode.

Specifically, we consider BRAMs in NV-FPGAs, which are essentially nonvolatile memory arrays distributed across NV-FPGA platforms. In addition to traditional storage function, we introduce computational function into BRAMs, referred to as computational BRAM (C-BRAM), and give architecture design as well as an operator allocation strategy to fully utilize high computing parallelism of C-BRAMs. We take neural network inference as an application instance to show benefits C-BRAMs could bring compared with existing NV-FPGAs.

To the best of our knowledge, this is the first work to study computational BRAM in the context of NV-FPGAs. We make the following contributions in this paper.

- We explore how to adapt NVM-based CiM architecture into BRAM in NV-FPGA to reduce area overhead of peripheral circuits.
- We develop a computational density aware operator allocation strategy to select appropriate resource type for operators, leveraging the high computational density of C-BRAM to its full potential.
- We integrate the proposed architecture and strategy into an open-source FPGA synthesis tool-chain for evaluation.

The remainder of this article is organized as follows. Section II summarizes related work. Section III presents detailed C-BRAM architecture design and operator allocation strategy. Section IV presents evaluation results and discussions. Finally, Section V gives conclusion.

II. RELATED WORK

In this section, we summarize related work on architecture design and synthesis flow for NV-FPGAs, as well as FPGAs with computation-in-memory.

A. Architecture and Synthesis of NV-FPGAs

Emerging NVMs are considered as replacements for current SRAMs in FPGA platforms, driven by their advantages of high density and low leakage power. A series of investigations have sought to align intrinsic characteristics of NVM with FPGA architectures through the block architecture redesign and the synthesis flows optimization. For instance, Chen et al. [5] leveraged 3D chip stacking technology to implement a PCM-based NV-FPGA platform, while Yazdanshenas et al. [3] proposed a 2-bank BRAM architecture for FPGA, adapting to the MTJ-RAM read/write circuits. Ju et al. [4] introduced a synthesis framework for adaptive data packing into SLC- or MLC-state BRAMs, and Huai et al. [2] proposed the lifetime-aware placement strategy, focusing on the remapping of frequently accessed data in nonvolatile BRAMs to enhance the lifetime of NV-FPGAs.

B. FPGAs with Computation-in-Memory

The CiM feature of nonvolatile arrays, such as ReRAM crossbars, has been extensively explored in the field of neural computation. Researchers have introduced the CiM blocks into programmable logic architectures to utilize this feature. For instance, Ji et al. [9] introduced FPSA, an architecture that integrates ReRAM-based crossbars, CLBs, and BRAMs into an array architecture, interconnected through routing resources. However, within this architecture, the CiM block serves exclusively for neural computation and is not intended for general-purpose storage. Similarly, Zha et al. [10] presented Liquid-Silicon, which employs NVM-based crossbar tiles for both sum-of-product logic and storage functions, effectively replacing traditional CLBs, BRAM, and routing resources. These works leverage the CiM capabilities of NVM to significantly enhance the performance of domain-specific applications. However, they require a comprehensive redesign of the FPGA-like programmable logic architecture.

There have been studies conducted to explore the existing BRAM for computation. Wang et al. [11] introduced neural cache technology into FPGA BRAMs, enhancing them with single-bit serial computing capability. Arora et al. [12] proposed CoMeFa, an architecture that facilitates single-bit serial computation within BRAM and offers ease of implementation. Chen et al. [13] presented BRAMAC, which supports both bit-serial and bit-parallel data processing and embeds control logic within BRAM. However, these studies primarily focus on SRAM-based FPGA platforms. In contrast, this paper uses the existing nonvolatile BRAM within NV-FPGA to achieve computational function, which is quite different from existing researches on SRAM-based FPGA in terms of both architecture and algorithms.

III. COMPUTATIONAL BRAM ON NV-FPGAS

In this section, we present the computational BRAM design in detail, including architecture design of C-BRAMs, and operator allocation strategy in NV-FPGAs.

A. Main Idea

The CiM has proved to achieve highly parallel computations, especially for neural network inference. However, this feature has not been well explored in NV-FPGAs yet. Thus in this paper, we aim to address the following questions. Firstly, how to adjust the BRAM architecture to integrate computational function into current area-efficient components to build C-BRAM. Secondly, how to fully utilize all computational components in NV-FPGAs, including CLBs, DSPs, and newly involved C-BRAMs, to achieve high computational density.

In Section III-B, we adapt typical CiM architecture of NVM into C-BRAMs. Specifically, we design a series of components, including a hybrid wordline driver, lightweight sense amplifier, and subneg & bypass module to enable BRAM with computational capability while minimizing area overhead. Without loss of generality, in this paper, we use ReRAM as an example to serve as material for building nonvolatile BRAMs.

In Section III-C, we introduce a computational density-aware operator allocation strategy. This strategy aims to achieve high-throughput in NV-FPGAs by guiding the mapping of computational operations onto computing resources like CLBs, DSPs, and C-BRAMs. Specifically, first, we establish area and performance cost models to estimate the area and performance costs of each operation under different computing resources. Then, we use a heuristic algorithm to guide the exploratory

search in solution space, aiming to find an optimal mapping solution that achieve the optimal computational density.

B. Architecture Design of C-BRAM

As shown in Figure 3 (a), we adapt a typical CiM architecture [7] to a dual-port 2-bank BRAM architecture [3]. The following changes have been made to bridge the architectural gap between CiM and BRAM.

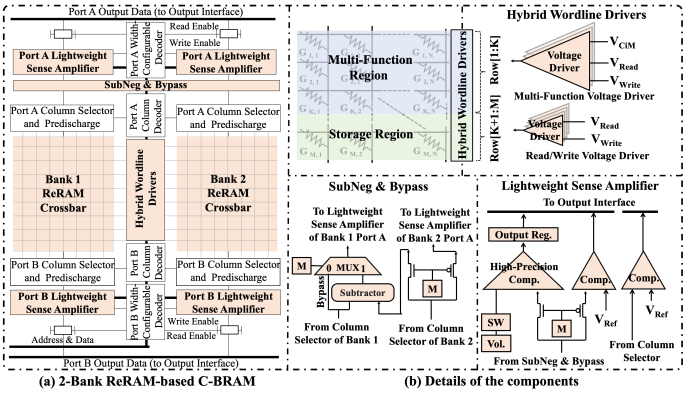

Fig. 3. Architecture of C-BRAM.

Firstly, we propose a hybrid wordline driver structure to reduce the number of multi-function voltage drivers and minimize area overhead. The multi-function voltage driver can provide the input voltage for computation or storage in the typical NVM-based CiM architecture, but it is more complex than read/write voltage driver in BRAM architecture. Using multi-function voltage drivers for each wordline will necessitate more drivers and additional input pins. This not only leads to a significant increase in the area of the BRAM itself, but also leads to routing congestion around the BRAM, resulting in increased routing area and potential performance issues. As illustrated in Figure 3 (b), we partition the crossbar into two discernible regions: the multi-function region, denoted as Row[1:K], and the storage region, denoted as Row[K+1:M]. The M signifies the total count of rows within the crossbar. A larger K indicates higher the computational parallelism, but also larger area cost. We set the value of K to the maximum bit width of storage mode that BRAM can accommodate, thus minimizing the modifications required for the input interface with an acceptable area overhead. Taking a 32Kb ReRAM-based BRAM as an example, this scheme saves 15 times the wordline driver area compared to the scheme using all multi-function voltage drivers i.e. $K = M$.

Secondly, we introduce a lightweight sense amplifier structure by trimming redundant functional circuits to reduce area overhead. In the CiM architecture, the sense amplifier consists of high precision comparator (Comp.) and functional circuits. The current indicating can be converted to digital level via the high precision comparator of sense amplifier. It supports CiM to work in either computation or storage mode, but with high area overhead. To minimize the area overhead in the lightweight sense amplifier structure, we specifically position the high-precision comparator on one port (Port A of Bank 1). Meanwhile, the remaining ports utilize regular comparators with smaller areas, which only support CiM to work in storage mode. In addition, the sense amplifier module of the CiM architecture can also integrate some functional circuits, such as ReLu and MaxPool [7]. In FPGAs, the CLBs and DSPs around BRAMs can be flexibly configured for various functions. For area consideration, we trim these functions to reduce area overhead while employing CLBs and DSPs in FPGA to implement these functions.

Finally, to fit the lightweight sense amplifier structure, we present the subneg & bypass module. It consists of a multiplexer, analog subtraction, and selection switch. It not only provides the capability for analog signal subtraction but also offers a flexible data path for signal transmission between bitlines of crossbar and lightweight sense amplifier structure.

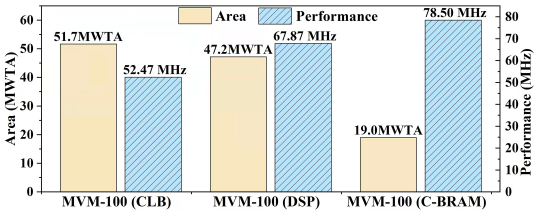

Fig. 4. Implementations of 8-bit 100x100 MVM with CLB, DSP, and C-BRAM.

As a new computational resource, C-BRAM provides another choice to execute specific operations (OPs). In order to test the performance of C-BRAM, we compare designs of a representative OP, i.e., 8-bit 100x100 MVM, whose computational logic is mainly implemented with CLB, DSP, and C-BRAM. As shown in Figure 4, C-BRAM delivers simultaneous area reduction and performance improvement compared with the other two designs. This confirms the potential of C-BRAM for computation-dense OPs. However, mapping all OPs to C-BRAM is obviously not a good solution. Thus we need to decide how to map OPs to computational resources, including CLB, DSP, and C-BRAM in NV-FPGAs, to fully utilize them and achieve system-level optimization.

C. Computational Density Aware OP Allocation

Here we focus on OPs that can be implemented on C-BRAMs. So we narrow down the definition of OPs to operators can be implemented by C-BRAMs, such as convolution OPs and MVM OPs. For others that C-BRAM does not support, we refer to them as non-targeted OPs, e.g., ReLu, BatchNorm, and control logic, which can be handled by existing mapping method.

The goal of OP allocation is to achieve high computational density, which is defined as the ratio of performance to occupied physical area. Thus we need to simultaneously optimize area and performance of the system. Area is the physical size the final implementation solution occupies, and performance can be evaluated by the length of critical path in the solution. However, neither of them can be accurately derived at early-stage of the synthesis flow. Therefore, we

use resource consumption to estimate area and inter-block connections to assess performance.

There may be hundreds or thousands of OPs for allocation in a large-scale application, the solution space is quite large. Blindly reducing the area can lead to increased routing congestion, resulting in degraded performance, while the opposite can lead to increased area. As shown in Algorithm 1, we employ simulated annealing (SA) to find an optimal mapping solution that satisfies the optimal computational density. SA is a heuristic algorithm that uses random search to explore the solution space and uses probabilistic acceptance to gradually converge to the optimal solution [14].

Algorithm 1 CD-Aware OP Allocation Strategy

Require: OPs List, non-targeted OPs List;
Ensure: Mapping Solution;
1: Evaluate resource consumption Res_{NonOP} and connection Con_{NonOP} of all non-targeted OPs;
2: $Res_{Ava} = Res_{Total} - Res_{NonOP}$;
3: $Con_{Cur} = Con_{NonOP}$;
4: Evaluate resource consumption $R_{[OP_i.ET_j]}$ and connection $C_{[OP_i.ET_j]}$ for each execution type ET of each OP;
5: **for** OP_i in OPs List **do**
6: Randomly select an execution type ET_{Sel} for OP_i;
7: $OP_i.ET_{[Cur]} \Leftarrow ET_{Sel}$;
8: $Con_{Cur} = Con_{Cur} + C_{[OP_i.ET_{[Cur]}]}$;
9: $Res_{Ava} = Res_{Ava} - R_{[OP_i.ET_{[Cur]}]}$;
10: **end for**
11: $T \Leftarrow T_{start}$;
12: **while** $T \leq T_{end}$ **do**
13: Randomly select an OP_i from OP List;
14: **for** each used execution type ET **do**
15: $AR[ET] = \frac{Res_{Ava}[ET]}{Res_{Total}[ET]}$;
16: **end for**
17: Select an ET_{Sel} with probability $\frac{AR[ET_{sel}]}{\sum AR[ET]}$;
18: $Cost_{Cur} = Con_{Cur}$;
19: $Cost_{Try} = Con_{Cur} - C_{[OP_i.ET_{[Cur]}]} + C_{[OP_i.ET_{[Sel]}]}$;
20: $\Delta C = Cost_{Try} - Cost_{Cur}$;
21: **if** $random(0,1) < exp(-\Delta C/T)$ **then**
22: $Con_{Cur} = Cost_{Try}$;
23: $Res_{Ava} = Res_{Ava} + R_{[OP_i.ET_{[Cur]}]}$;
24: $OP_i.ET_{[Cur]} = ET_{Sel}$;
25: $Res_{Ava} = Res_{Ava} - R_{[OP_i.ET_{[Sel]}]}$;
26: **end if**
27: $T = \gamma T$;
28: **end while**

Before allocation, we first collect some basic information of both FPGA platform and OPs. Since non-targeted OPs are not involved in the proposed allocation strategy, we first evaluate resources needed by non-targeted OPs, and exclude them from the platform. The remaining resource, noted as Res_{Ava}, is available for targeted OPs (Lines 1-2). And connection Con_{Cur} is initialized with connection status with non-targeted OPs being allocated (Line 3). Then for each OP, we evaluate its cost in terms of resource and connection if it is mapped to a certain computation resource type, i.e., CLB, DSP, or C-BRAM (Line 4). This information is helpful to figure out beneficial allocation in the following SA steps.

At the beginning of SA, each OP is randomly allocated to resource type (Line 6), with resource and connection information updated accordingly (Lines 7-9). Then it comes with iterative change-and-evaluate steps (Lines 12-28). In each iteration, an OP is randomly selected and examined whether to change the allocation or not. The main idea is to choose a new resource type, and try to re-allocate the OP to the new type. The re-allocation is decided to be accepted or not according to benefit brought by the change (Lines 18-21). When choosing a new resource type, area is considered for optimization. Thus, instead of random selection, we use available resource as the criteria, where the type with higher available ratio would have bigger chance to be selected (Lines 14-17). When deciding to accept the re-allocation or not, performance is evaluated. The costs before and after re-allocation, noted as $Cost_{Cur}$ and $Cost_{Try}$, respectively, are compared. In Line 21, the condition accepts solutions with negative ΔC, i.e., better performance, but it is also possible to accept solutions with some possibility when ΔC is positive, i.e., with worse performance.

The temperature plays an important role during SA. At first, with comparatively high temperature, most re-allocations would be accepted since acceptance condition in Line 21 is easily achieved. This leads to area-oriented optimizations because the resource type with high available ratio is selected (Line 17). As iteration goes on, the temperature is decreased according to a discount factor γ (Line 27). The acceptance condition is harder to be satisfied and re-allocations with better performance are more likely to be accepted. This leads to performance optimization. The algorithm terminates when the temperature T falls below the preset final temperature T_{end} (Line 12). Thus the whole algorithm first explores area-oriented solutions and then fine-tunes allocations to achieve better performance. Finally, the allocation solution with high computation density is obtained.

The proposed allocation strategy determines resource type for each OP. The decision can be specified at the hardware description language (HDL) level and fed into synthesis flow to generate configuration bit-stream file runnable at NV-FPGA platforms. It needs to be mentioned that, for those OPs whose types are specified as C-BRAM, we need to add corresponding implementation of C-BRAM to include them into the synthesis flow. In this way, they can go through the following steps such as packing, placement, and routing, together with non-targeted OPs.

IV. Evaluation

In this section, we introduce experimental setup, report evaluation results and give discussions.

A. Experimental Setup

The proposed C-BRAM architecture and computational density aware (CD-Aware) OP allocation strategy have been integrated into VTR [15], an open-source FPGA synthesis tool-

chain[1]. C-BRAM is defined as a new working mode of BRAM, which can be handled through traditional physical synthesis flow. We compare the following implementations in terms of computational density.

- **SRAM-FPGA**. Traditional SRAM-based FPGA architecture with default synthesis flow [15].
- **SNV-FPGA**. NV-FPGA with SRAMs in BRAM being replaced with ReRAM. BRAMs are only used for storage. Default synthesis flow is applied.
- **CNV-FPGA (w/o CDA)**. NV-FPGA with the proposed C-BRAM. But the proposed CD-aware OP allocation strategy is not applied. All OPs are blindly allocated to C-BRAMs.
- **CNV-FPGA (w/ CDA)**. NV-FPGA with the proposed C-BRAM, and the proposed CD-aware OP allocation scheme is used.

Representative neural networks including LeNet, LSTM, and CNNs are used to evaluate the proposed architecture and strategy. Details are shown in Table I.

TABLE I
INFORMATION OF BENCHMARK

Benchmark	Domain	# OPs
LSTM [16]	natural language processing	8
LeNet [17]	computer vision	1986
CNN48_55	image data processing	48
CNN48_33	image data processing	48
CNN48_M[2]	image data processing	48

B. Evaluation Results

Figure 5 shows the computational density of all tested implementations, where data are normalized to those of SRAM-FPGA. Compared to SRAM-FPGA, SNV-FPGA achieves a 6% increase in computational density. This is due to the higher storage density brought by NVM. CNV-FPGA (w/o CDA) exhibits improved computational density for LSTM, LeNet, and CNN48_55 benchmarks, while experiencing a decrease for CNN48_33 and CNN48_M. This variation arises from the fact that not all OPs are suitable for allocation to C-BRAM. CNV-FPGA (w/ CDA) achieves the highest computational density for all benchmarks, i.e., 124% improvement over SRAM-FPGA on average. This is because the proposed CD-aware allocation strategy is able to allocate OPs to appropriate type of computation resource, and thus effectively enhances the resource utilization and reduces routing congestion.

In order to disclose more details, we take CNN48_55 as an example to show its solution space of OP allocation, as depicted in Figure 6. CNN48_55 has 48 OPs of 5x5 convolution kernel, where each can be allocated onto CLB, DSP, or C-BRAM. Group ID in Figure 6 indicates the number of OPs that have been mapped to C-BRAM. For example, in Group 0, there are no OPs mapped to C-BRAM, these OPs are mapped

[1]We modify the latency and area based on the Stratix-10-like architecture to support NV-FPGAs and incorporate the allocation strategy before logical synthesis.

[2]CNN48_M includes multiple types of convolution OPs, with 24 5x5 convolution kernels and 24 3x3 convolution kernels.

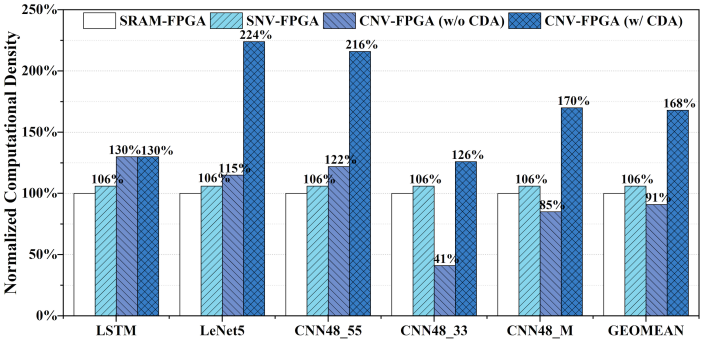

Fig. 5. Computational density improvement of CNV-FPGA.

to the CLBs and DSPs. In the subfigure, the leftmost point represents all OPs are mapped to DSPs, and the rightmost point indicates all OPs are mapped to CLBs. All kinds of possible allocation solutions are enumerated in this figure. It can be observed that, as more OPs are re-allocated from DSP to CLB, the computational density roughly first increases and then decreases. Generally, if one convolution OP is allocated to CLB, it tends to consume larger area than DSP since it needs multiple CLBs for logic implementation with more complicated routing. Thus allocating all OPs to CLB delivers much lower computational density than DSP.

In terms of inter-group comparison, with more OPs allocated to C-BRAM, the maximum computational density in each group initially increases and then decreases. Thus it is beneficial to re-allocate OPs to C-BRAM to exploit its high parallelism. However, as more C-BRAMs are occupied, BRAM becomes the bottleneck, resulting in an increase in physical area and thus lower computational density.

As shown in Figure 6, the proposed CD-aware OP allocation strategy achieves a computational density close to the global optimum. This confirms the efficacy of the proposed strategy in exploring the solution space and achieving a favorable balance between area and performance.

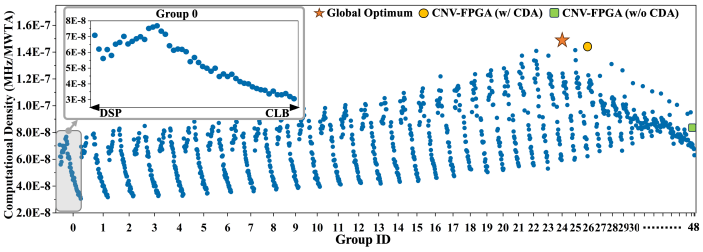

Fig. 6. The solution space of CNN48_55.

C. Discussions

In this section, we set up more evaluations to explore performance of the proposed strategy with various scales and precisions. We also evaluate the area overhead of C-BRAM and accuracy of neural network models implemented in NV-FPGAs

1) With various scales of OPs: In order to compare OPs with various scales, we choose two kinds of basic modules widely

adopted in neural networks, i.e., convolution and general matrix multiplication, for evaluation. For convolution, we implement CONV3x3, CONV5x5, CONV7x7. And for general matrix multiplication, we implement GEMM32, and GEMM64. For each OP, we measure the computational density when it is allocated in SRAM-FPGA, and C-BRAM in CNV-FPGA, respectively. As shown in Figure 7, the computational density improvement CNV-FPGA achieves is 45% for CONV3x3-6, 370% for CONV5x5-6, and 398% for CONV7x7-6, respectively. This confirms the superiority of C-BRAM when implementing large-scale OPs. Similar conclusions can be drawn for GEMM.

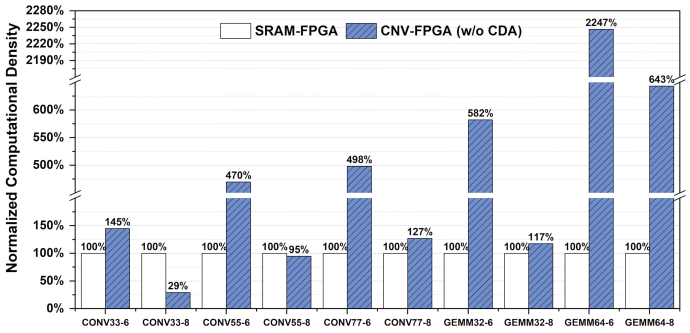

Fig. 7. Computational density of OP implementations with various scales and precisions.

2) With various precisions of weights in OP: C-BRAM supports various precisions of weights in OPs. Labels "*-6" and "*-8" in Figure 7 represent 6-bit and 8-bit precision of weights, respectively. It can be seen that computational density of 8-bit precision is always lower than the 6-bit one. It is because that computation on higher-precision weights consumes more C-BRAMs than lower-precision ones. In CONV3x3-8 and CONV5x5-8, implementations on CNV-FPGA are even worse than SRAM-FPGA. The reason is that allocating small-scale high-precision convolution to C-BRAMs, it needs comparatively large number of C-BRAMs to satisfy the precision requirement, while the utilization of C-BRAMs is quite low. This leads to a large area of the whole implementation, and thus unsatisfactory computational density. Note that Figure 7 show results of CNV-FPGA (w/o CDA). When the proposed CD-aware allocation strategy is applied, these kinds of OPs will be allocated to DSPs instead of C-BRAMs.

3) Area overhead of C-BRAM: In the proposed C-BRAM architecture, additional peripheral circuit components are required to support computation. We have evaluated the area cost of C-BRAM by COFFE [18], where the area parameters of high precision comparator, multi-function voltage driver, and subtractor are referenced to MNSIM [19]. The proposed area-efficient C-BRAM architecture occupies only 47% of the area compared to using original peripheral design of CiM.

4) Model accuracy on CNV-FPGAs: Due to analog computations and signal conversions, there is a precision loss. We test accuracy of LeNet with 6-bit weights on CIFAR-10 dataset with MNSIM [19], where the parameters of the sense amplifier, wordline driver, and crossbar are set according to C-BRAM architecture. Compared with full-precision implementation in DSPs, the accuracy loss is tested to be 2%. In reality, as a trade-off, precision can be determined according to accuracy requirement of AI applications.

V. Conclusion

In this paper, we study computational BRAM design in NV-FPGA platforms. First, we present the architecture design to integrate computational function into nonvolatile BRAM while minimizing area overhead. Furthermore, to smartly map operators to different kinds of computation resources in NV-FPGA, we propose a computational density aware operator allocation strategy, which simultaneously optimizes the area and performance of implementations. We take neural network inference as an application to show benefits C-BRAM brings. Evaluation results demonstrate a 68% and 62% improvement in computational density compared to traditional SRAM-based FPGA and existing NV-FPGA, respectively.

References

[1] www.xilinx.com/products/silicon-devices/fpga.html.

[2] S. Huai, W. Song, M. Zhao, *et al.*, "Performance-aware Wear Leveling for Block RAM in Nonvolatile FPGAs," in *DAC*, pp. 1–6, ACM, 2019.

[3] S. Yazdanshenas, K. Tatsumura, and V. Betz, "Don't forget the memory: Automatic block ram modelling, optimization, and architecture exploration," in *FPGA*, pp. 115–124, ACM, 2017.

[4] L. Ju, X. Sui, S. Li, *et al.*, "NVM-based FPGA block RAM with adaptive SLC-MLC conversion," *IEEE TCAD*, vol. 37, no. 11, pp. 2661–2672, 2018.

[5] Y. Chen, J. Zhao, and Y. Xie, "3D-NonFAR: Three-dimensional non-volatile FPGA architecture using phase change memory," in *ISLPED*, pp. 55–60, ACM/IEEE, 2010.

[6] H. Zheng, H. Zhang, S. Xu, *et al.*, "Adaptive mode transformation for wear leveling in nonvolatile FPGAs," *IEEE TCAD*, vol. 41, no. 11, pp. 3591–3601, 2022.

[7] P. Chi, S. Li, C. Xu, *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, pp. 27–39, ACM, 2016.

[8] Z. Shen, J. Wu, X. Jiang, *et al.*, "PRAP-PIM: A weight pattern reusing aware pruning method for ReRAM-based PIM DNN accelerators," *Elsevier HCC*, vol. 3, no. 2, pp. 100–123, 2023.

[9] Y. Ji, Y. Zhang, X. Xie, *et al.*, "FPSA: A full system stack solution for reconfigurable reram-based nn accelerator architecture," in *ASPLOS*, pp. 733–747, ACM, 2019.

[10] Y. Zha and J. Li, "Liquid silicon-monona: A reconfigurable memory-oriented computing fabric with scalable multi-context support," in *ASPLOS*, pp. 214–228, ACM, 2018.

[11] X. Wang, V. Goyal, J. Yu, *et al.*, "Compute-Capable Block RAMs for Efficient Deep Learning Acceleration on FPGAs," in *FCCM*, pp. 88–96, IEEE, 2021.

[12] A. Arora, T. Anand, A. Borda, *et al.*, "CoMeFa: Compute-in-Memory Blocks for FPGAs," in *FCCM*, pp. 1–9, IEEE, 2022.

[13] Y. Chen and M. S. Abdelfattah, "BRAMAC: Compute-in-BRAM Architectures for Multiply-Accumulate on FPGAs," in *FCCM*, pp. 52–62, IEEE, 2023.

[14] P. J. M. Laarhoven and E. H. L. Aarts, "Simulated Annealing: Theory and Applications," *1987 Simulated Annealing*.

[15] J. Luu, J. Goeders, M. Wainberg, *et al.*, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *ACM TRETS*, vol. 7, no. 2, pp. 1–30, 2014.

[16] A. Arora, A. Boutros, D. Rauch, *et al.*, "Koios: A Deep Learning Benchmark Suite for FPGA Architecture and CAD Research," in *FPL*, pp. 355–362, IEEE, 2021.

[17] Y. LeCun, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[18] S. Yazdanshenas and V. Betz, "COFFE 2: Automatic modelling and optimization of complex and heterogeneous FPGA architectures," *ACM TRETS*, vol. 12, no. 1, pp. 1–27, 2019.

[19] L. Xia, B. Li, T. Tang, *et al.*, "MNSIM: Simulation platform for memristor-based neuromorphic computing system," *IEEE TCAD*, vol. 37, no. 5, pp. 1009–1022, 2017.