A Runtime Reconfigurable Design of Compute-in-Memory based Hardware Accelerator

Anni Lu, Xiaochen Peng, Yandong Luo, Shanshi Huang, Shimeng Yu
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, USA
shimeng.yu@ece.gatech.edu

***Abstract*—Compute-in-memory (CIM) is an attractive solution to address the "memory wall" challenges for the extensive computation in machine learning hardware accelerators. Prior CIM-based architectures, though can adapt to different neural network models during the design time, they are implemented to different custom chips. Therefore, a specific chip instance is restricted to a specific network during runtime. However, the development cycle of the hardware is normally far behind the emergence of new algorithms. In this paper, a runtime reconfigurable design methodology of CIM-based accelerator is proposed to support a class of convolutional neural networks running on one pre-fabricated chip instance. First, several design aspects are investigated: 1) reconfigurable weight mapping method; 2) input side of data transmission, mainly about the weight reloading; 3) output side of data processing, mainly about the reconfigurable accumulation. Then, system-level performance benchmark is performed for the inference of different models like VGG-8 on CIFAR-10 dataset and AlexNet, GoogLeNet, ResNet-18 and DenseNet-121 on ImageNet dataset to measure the tradeoffs between runtime reconfigurability, chip area, memory utilization, throughput and energy efficiency.**

Keywords—convolutional neural network, hardware accelerator, compute-in-memory, reconfigurable architecture

I. Introduction

State-of-the-art convolutional neural network (CNN) based learning algorithms have achieved remarkable success in various artificial intelligence applications such as computer vision, speech recognition and language translation, etc. [1]. However, due to the requirement of high bandwidth and power consumption for data movement, computing platforms with traditional von Neumann architecture are inadequate for efficient calculation of CNNs. Compute-in-memory (CIM) is a promising solution to alleviate the memory access bottleneck by configuring the memory arrays for dot-product computation in a parallel fashion by summing up the column currents. With recent progress in emerging non-volatile memory (eNVM) devices such as resistive random access memory (RRAM) [2], phase change memory (PCM) [3], and ferroelectric field-effect transistor (FeFET) [4], the application of CIM-based CNN accelerator is even more attractive since eNVMs offer low leakage power and non-volatility that are necessary for dynamic power-gating and instant on and off operations in edge devices.

Besides the computation efficiency, flexibility is another important factor to be considered for the choice of computing platforms. The CNN-based learning algorithms are being rapidly developed in recent years. Since AlexNet [5], one of the pioneering works proposed in 2012, many new models such as VGG [6] in 2014, GoogLeNet [7] in 2014, ResNet [8] in 2015, DenseNet [9] in 2016, MobileNet [10] in 2017 and EfficientNet [11] in 2019 are explosively emerging, not to mention the countless fine-tuned or improved variants of these networks for specific applications. However, the development cycle of an application-specific integrated circuit (ASIC) design can be half to two years, which is hard to follow up the rate of new algorithm designs. This is one of the reasons why FPGA is becoming so popular for prototyping emerging algorithms nowadays.

Prior CIM-based architectures [12]-[14], though can adapt to different neural network models during the design time, it should be clarified that *though these architectures were typically evaluated using various models, in fact each model was implemented on a separate custom chip with different on-chip resources but following the same principle of chip architecture*. To our best knowledge, the runtime reconfiguration of a pre-fabricated chip instance is largely unexplored for CIM-based architectures. In this paper, we will investigate the feasibility of runtime reconfiguration in CIM-based architectures to support different models. To enable this, some reconfigurable circuit modules are needed, and a flexible dataflow and weight mapping strategy is required. One particular challenge is that sometimes the model size is larger than the on-chip memory capacity, thus the weight reloading scheme needs to be considered. We will also explore eNVM-based CIM designs, which hold advantages than FPGA not only in throughput and energy efficiency but also nonvolatility which makes it possible to instantly start inferencing without initializing the model when powered on.

In this paper, a runtime reconfigurable CIM-based accelerator is designed and benchmarked with different CNN algorithms. First, mapping method and additional peripheral circuits to support the reconfigurability are discussed. Then, system-level performance benchmark is performed using modified DNN+NeuroSim framework [15] with a wide range of network models with very different sizes, assuming a pre-fabricated chip with a fixed capacity of FeFET, one of the most promising emerging technologies for edge inference. The trade-offs of runtime reconfigurability are discussed as it introduces overhead of chip area and degradation of memory utilization, throughput and energy efficiency.

II. Background

A. Convolutional Neural Network (CNN) Basics

CNN consists of multiple convolutional layers to learn the salient features and a few fully-connected layers for classification. In this paper, we focus on the acceleration of the inference engine where the weights have been pre-trained offline. In a convolutional layer, an output feature map (OFM) is the result of multiply-and-accumulation (MAC) operations on a collection of weights (or filters) operating in a sliding window fashion over the input feature map (IFM). Consider the case where the IFM of size W×W×D is processed by N filters, each of size K×K×D. Then the OFM of size W×W×N is computed as follows. Here, I, W, O are the IFM, weights and OFM, respectively.

$$O[x][y][n] = \sum_{i=0}^{K-1}\sum_{j=0}^{K-1}\sum_{k=0}^{D-1} I[x+i][y+j][\mathrm{k}] \times \mathrm{W}[i][j][\mathrm{k}][\mathrm{n}]$$

B. FeFET-based CIM Basics

FeFET is a promising device technology with tunable threshold voltage by partially flipping the ferroelectric domains in the gate stack. Compared to other eNVM technologies, FeFET offers higher cell resistance (i.e., a few hundred kΩ [4]), thus better energy-efficiency. FeFET has been successfully demonstrated in industrial 28nm platform [4]. FeFET may need a normal transistor that is controlled by word line (WL) to select individual FeFET during the row-by-row programming mode, while the WLs are all turned on during the parallel read-out mode. Read voltages are applied to RS (read select). Currents are sensed from SL (source line) as the multiplication results of input voltages and cell conductance as Fig. 1(a) shows. As shown in Fig. 1(b), the synaptic subarray also contains peripheral circuits such as switch matrix (to select specific rows or columns) and its drivers, MUX and its decoder, analog-to-digital converter (ADC) and shift-add to support multi-bit input and multi-bit weight operations.

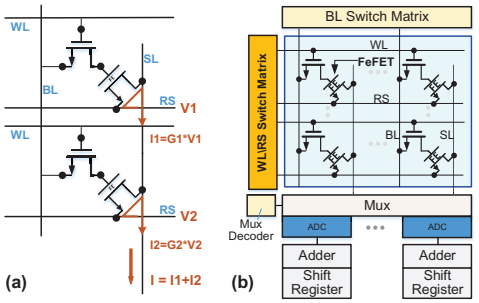

Fig. 1. (a) Weighted sum operation in FeFET-based synaptic cells. (b) FeFET-based synaptic array with peripheral circuitry.

III. Reconfigurable Design Methodologies

A. Reconfigurable Mapping Methods

The generic architecture of CIM-based CNN accelerator design is shown in Fig. 2. The architecture has a hierarchy of chip, tile, processing element (PE) and subarray from top to down. Each level contains its digital peripheries, including input/output buffers, interconnects (based on H-tree routing), accumulation, activation and pooling units .

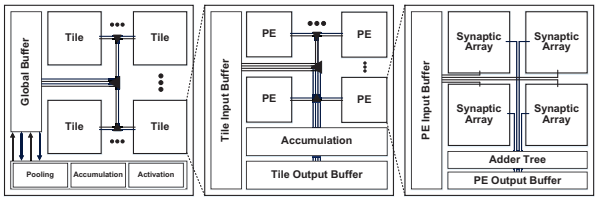

Fig. 2. The hierarchical diagram of CIM-based architecture.

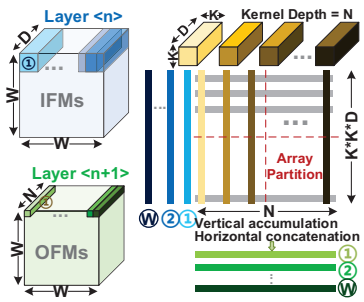

Fig. 3. A conventional mapping (CM) method of input and weight data, with 3D kernel unrolled into a long column vector.

With the CIM approach, the kernels (weights) will be mapped into memory arrays as conductance of each cross-point. Since the partial sums in each 3D kernel will be summed up to get the final output, it is straightforward to unroll each 3D kernel into a long vertical column, by utilizing the nature of crossbar array which could perform the sum of dot-products in the SLs. Fig. 3 shows such weight mapping method [16], and we refer it as conventional mapping (CM) in this paper. We also consider a novel mapping (NM) method [17] to fully reuse the input data as shown in Fig. 4. In this way, K×K PEs are needed for the K×K kernels. As the kernel slides over the inputs, the IFMs can be reused and transferred from the neighboring PEs. Therefore, the new IFM will only be sent to K PEs and the data movement between buffer and memory arrays is reduced by K times. But obviously, this NM method is not applicable for convolutional layers with 1×1 kernel size and fully-connected layers, so CM method is still utilized in these layers. If more than one tiles of memory arrays are required to map one layer of weights, a mapping rule should be followed that *the output of each tile should be accumulated vertically and concatenated horizontally*. This rule applies for both CM and NM methods, because the input channel D is distributed in different rows and the output channel N is mapped along the columns. For a custom chip design, NM is typically preferred over CM to improve the throughput and energy efficiency [15].

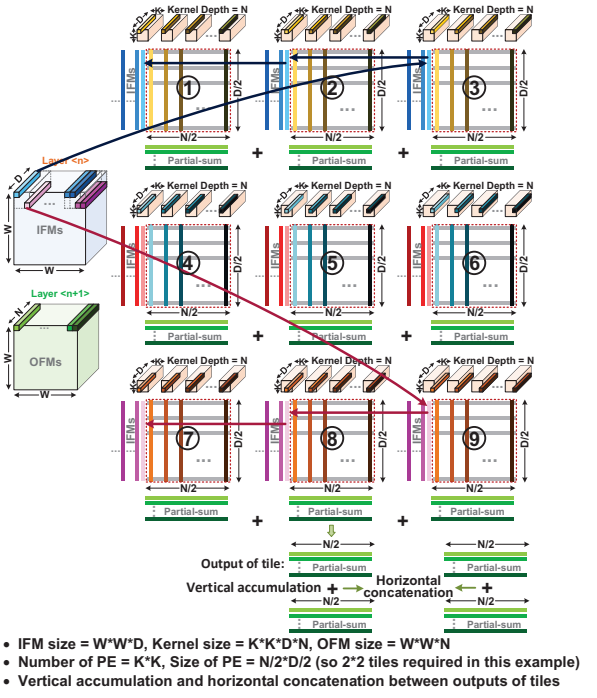

Fig. 4. A novel mapping (NM) method that the weights are mapped along the spatial location to a group of PEs to fully reuse IFMs.

Without losing generality, we will support tile-level reconfigurable mapping in this design. One layer of weights occupies at least one tile of memory arrays. CM can be easily applied to different networks with distinct kernel size, channel or depth. However, it could be a challenge for NM because different numbers of PE are desired by distinct kernel sizes, which may not match the predefined chip configuration. To solve this problem, we consider a reconfigurable NM method on the following three scenarios:

1) Same desired number of PE as predefined, while the desired size of PE is different from the predefined one.

a) If the desired size of PE is smaller, the unused memory arrays can be utilized by weight duplication for speed up.

b) If the desired size of PE is larger, then each PE has to be partitioned into different tiles. Fig. 5 shows an example that the desired PE size is four times of the predefined one. The original operation is accumulating the results of all the 9 PEs. After segmentation, still, the 9 PEs inside each tile will be accumulated, and the output of the tiles will be added vertically and combined horizontally following the mapping rule, so that the computation is actually the same as before.

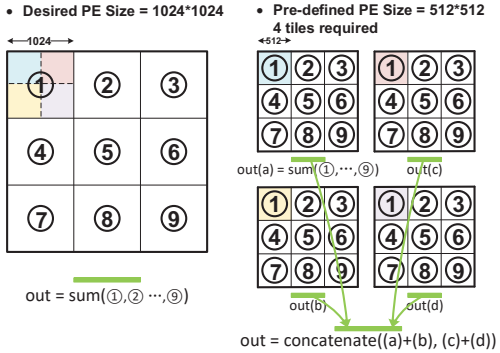

Fig. 5. Mapping example with a larger desired PE size than the pre-defined one.

2) Same desired size of PE as predefined, while the desired number of PE is different from the predefined one.

a) If the desired number of PE is smaller, then the weights originally mapped in the next rows of tiles can be brought forward since the output of tiles should be accumulated vertically. For tiny layers with only one row of tiles, duplicated weights could be employed to accelerate the computation and avoid the waste of memory. Fig. 6 is an example of 2×2 kernel size while mapped with 3×3 predefined number of PE.

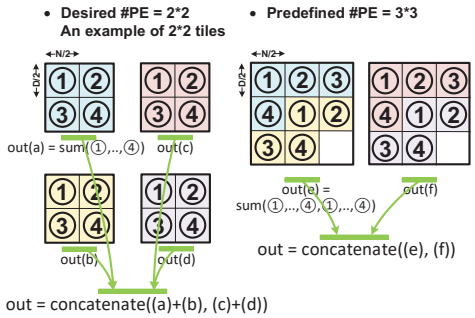

Fig. 6. Mapping example with a smaller desired #PE (2×2) than the pre-defined one (3×3).

b) If the desired number of PE is larger, then the original one tile might be split into several tiles. To comply with the mapping rule that the output of tiles is added vertically and combined horizontally, the separated tiles should be placed in the same column. Fig. 7 is an example of 4×4 kernel size while mapped with 3×3 predefined number of PE.

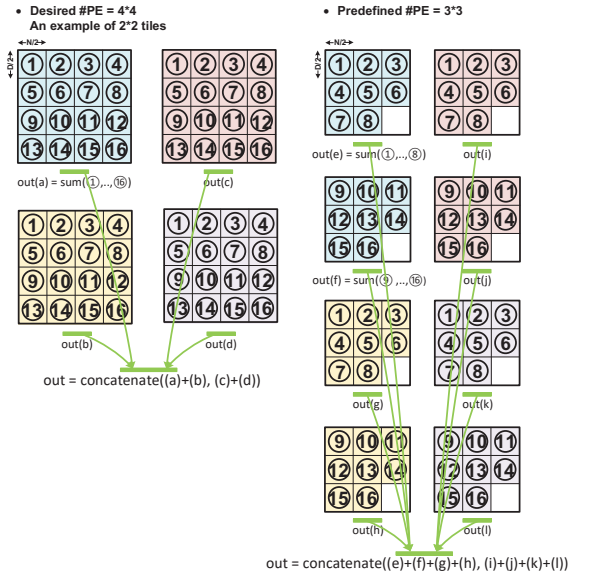

Fig. 7. Mapping example with a larger desired #PE (4×4) than the pre-defined one (3×3).

3) The desired number and size of PE are both different from the predefined ones. This is an integration of the above two scenarios and it will follow the strategy shown in the pseudo-code below:

$$\#row\_of\_tile = \frac{input\_channel}{predefined\_PE\_size}$$

$$\#col\_of\_tile = \frac{kernel\_depth}{predefined\_PE\_size}$$

$$\#row\_of\_tile = \left\lceil \#row\_of\_tile \times \frac{(desired\_PE\_num)^2}{(predefined\_PE\_num)^2} \right\rceil$$

if desired_PE_size < predefined_PE_size

$$Duplicate \left\lfloor \frac{predefined\_PE\_size}{desired\_PE\_size} \right\rfloor^2 times$$

if desired_PE_num < predefined_PE_num

$$Duplicate \left\lfloor \frac{predefined\_PE\_num}{desired\_PE\_num} \right\rfloor^2 times$$

B. *Input Side of Data Transmission*

When doing the calculation, IFM should be transferred from the global buffer to PE-level buffer and finally sent to the ports of the corresponding memory sub-arrays. Compared with custom design, the reconfigurable chip with constrained area might be insufficient for some large or deep networks, then the weight reloading would be necessary [18]. Therefore, the input side of data transmission should support for both IFM and weights transfer.

Taking a tile with 3×3 PEs, each PE with 4×4 sub-arrays, and each sub-array with 128×128 memory cells as an example. The input data is shared between columns of sub-arrays, because the MAC operation should be performed with IFM and each depth of kernel. If given that the calculations for all the rows of all the sub-arrays are performed in parallel, then 128×4 bits of IFM would be required for one-cycle computation of each PE. Thus, the PE-level buffer capacity should be at least 128×4 bits as the last level of buffer. The PE buffer interface width and bus width are also configured to 128×4 bits to match the buffer size. Then in the tile level, a group of multiplexers will determine which PE is the destination of the 128×4 bits. This is the input data transmission custom design rule.

For reconfigurable design, the input side buffer and bus should be able to transfer the weights for reloading. Although the parallel reading (computation) of a memory array is feasible, the writing (programming) of the weights must be operated row by row. Hence, if the programming of different sub-arrays is performed in parallel and the memory device supports 4-bits per cell as an example, the required weight bits for one PE should be 128×4×4×4, and for one tile 128×4×4×4×9 bits are required. The capacity of PE buffer has to be enlarged to 128×4×4×4 bits, but the tile bus width remains at 128×4 bits because an extremely wide global bus is unaffordable. Two stages of multiplexers are configured in the tile level as shown in Fig. 8(a): the first stage chooses which cell bit, and the second one decides the destination of which columns of sub-arrays inside which PEs. Besides the required bits, another difference is that IFM is given as the turn-on/off voltage on Read Select (RS), while weights are programmed into the memory cells from Bit Line (BL). It means the bus connected from PE buffer to sub-arrays cannot be shared for IFM and weight, as a result, additional multiplexers inside PEs are required to select between input or weight bus as in Fig. 8(b).

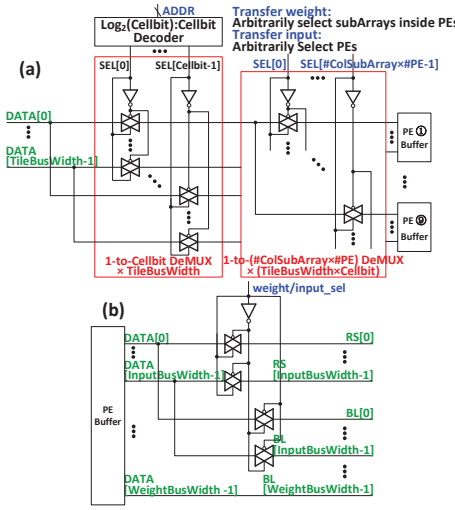

Fig. 8. (a) Tile-level (b) PE-level input buffer and bus. InputBusWidth = subArrayRow×#RowSubArray = 128×4; WeightBusWidth = InputBusWidth×Cellbit×#ColSubArray = 128×4×4×4; TileBusWidth = 128×4; PEBufferWidth = 128×4×4×4.

C. Output Side of Data Accumulation

For the output side, design effort is mainly on the reconfigurable accumulation. The reconfigurable mapping rule compatible with different kernel sizes leads to the accumulation of an arbitrary number of PEs inside a tile. Also, the inter-tile accumulation is different for different networks. Therefore, a module of reconfigurable accumulator and router between tiles for global accumulations is required.

The basic accumulator for custom design is implemented by the adder tree. To realize the runtime reconfiguration, for each adder inside the adder tree, a demultiplexer at each input and a multiplexer at the output are required for bypass selection as shown in Fig. 9. For the global accumulation, taking a chip with 6×6 tiles mapped with VGG-8 as an example. Although in Section 3.1 we investigated the required rows and columns of tiles for each layer, the real location does not necessarily match the dimensions as long as the rule that accumulating tiles vertically and concatenating them horizontally is followed, so that the potential memory waste caused by the irregular shapes of each layer could be avoided. As shown in Fig. 10, the tiles are mapped along the columns in priority and the global accumulation is comprised by 6 groups of reconfigurable adder tree with 6 sets of input each. Each adder tree has the same structure as Fig. 9. The first set of input is selected from the first row of tiles and the output of the first adder tree, and so on for the next sets. The source from the output of the adder trees itself is aimed at supporting more cycles of accumulation. For VGG-8, the global accumulation is only required for layer 7 and the output of every 6 tiles should be accumulated. The multiplexers help with the routing of the input and can decrease the required cycles for accumulation.

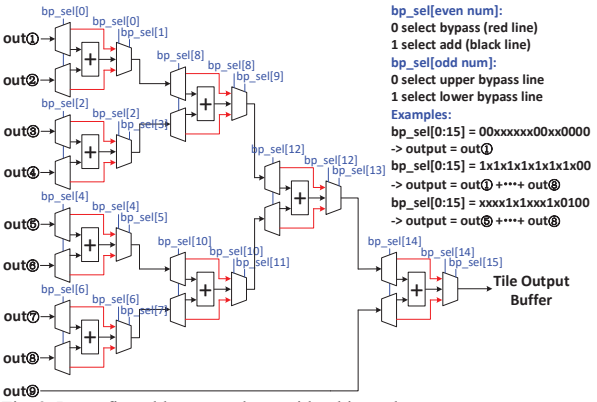

Fig. 9. Reconfigurable accumulator with arbitrary bypass.

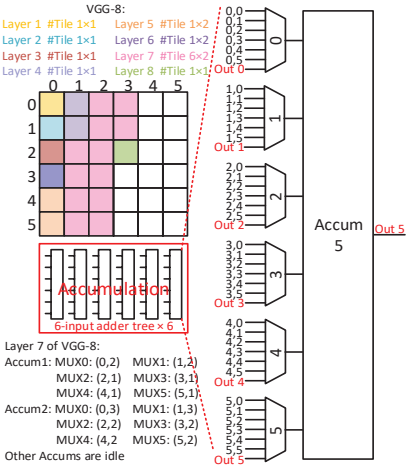

Fig. 10. Global accumulation with flexible input. Taking the mapping of VGG-8 as an example.

D. Other Design Considerations

There are some other design considerations to support the runtime reconfigurability. Firstly, the global buffer is constrained to a predefined size and might be inadequate for some large networks. Comparably large capacity is reserved in case, and data will be transferred to DRAM if exceeding its capacity, which will cause additional latency and large energy consumption. Secondly, the global accumulation has a predefined precision and might be insufficient for large layers that require too many cycles of accumulation. If so, the input would be truncated before the accumulation is operated. Thirdly, to support the compatibility of different models, both the ReLU and Sigmoid function for activation as well as both max pool and average pool for pooling layer are supported in the proposed reconfigurable design.

(a)

area breakdown (mm^2)		custom NM	custom CM	recfg
PE level	mux	0		0.0009
	buffer	**0.007**		**0.022**
	unit PE area	**0.098**		**0.115**
Tile level	mux	0.0003	0.0002	0.0165
	input hTree	0.009	0.009	0.010
	accum	**0.10**	0.03	**0.13**
	output hTree	**0.17**	0.07	**0.19**
	output buffer	0.008	0.005	0.008
	unit Tile area	**1.18**	0.50	**1.38**

(b)

VGG-8		custom	recfg
#tiles		NM×7+CM×18	21
Chip level area breakdown* (mm^2)	tiles	**17.27**	**29.00**
	global buffer	2.77	2.79
	global bus	9.90	9.00
	global accum	0.18	0.14
	other	0.05	0.94
	chip area	**30.17**	**41.87**
memory utilization		**93.47%**	**20.23%**
throughput(FPS)		3205.31	2979.44
energy efficiency(TOPS/W)		29.52	27.03

(c)

ResNet-18		custom	recfg
#tiles		NM×20+CM×3	23
Chip level area breakdown* (mm^2)	tiles	**25.09**	**31.77**
	global buffer	4.27	4.27
	global bus	9.11	9.85
	global accum	0	0.14
	other	0.72	0.94
	chip area	**39.19**	**46.97**
memory utilization		**87.08%**	**33.31%**
throughput(FPS)		471.44	456.36
energy efficiency(TOPS/W)		13.04	11.34

(d)

GoogLeNet		custom	recfg
#tiles		NM×23+CM×40	49
Chip level area breakdown (mm^2)	tiles	47.17	67.67
	global buffer	4.26	8.46
	global bus	60.53	20.99
	global accum	0	0.27
	other	1.14	1.32
	chip area	113.10	98.71
memory utilization		**57.74%**	**33.56%**
throughput(FPS)		307.76	311.55
energy efficiency(TOPS/W)		**10.97**	**7.76**

(e)

AlexNet		custom	recfg
#tiles		NM×4+CM×115	49
Chip level area breakdown (mm^2)	tiles	62.34	67.67
	global buffer	3.22	8.46
	global bus	114.33	20.99
	global accum	0.39	0.27
	other	0.10	1.32
	chip area	180.38	98.71
memory utilization		**97.96%**	**55.43%**
throughput(FPS)		1754.44	1895.59
energy efficiency(TOPS/W)		**22.72**	**13.31**

(f)

DenseNet-121		custom	recfg
#tiles		NM×58+CM×64	49
Chip level area breakdown (mm^2)	tiles	100.47	67.67
	global buffer	17.01	8.46
	global bus	117.22	20.99
	global accum	0	0.27
	other	1.69	1.32
	chip area	236.39	98.71
memory utilization		**64.98%**	**47.71%**
throughput(FPS)		111.95	120.67
energy efficiency(TOPS/W)		**9.10**	**5.36**

Fig. 11. Area and performance comparison of custom and reconfigurable design. The reconfigurable chip has a fixed area≈100mm^2. (a) PE- and Tile- level area breakdown; (b) comparison based on VGG-8 on CIFAR-10, and the reconfigurable chip area reported is "used area" for fair comparison; (c) comparison based on ResNet-18 on ImageNet, and the reconfigurable chip area reported is "used area" for fair comparison; (d) comparison based on GoogLeNet on ImageNet, and weight reloading is required as the constrained 100 mm^2 is insufficient; (e) comparison based on AlexNet on ImageNet, and weight reloading is required as the constrained 100 mm^2 is insufficient; (f) comparison based on DenseNet on ImageNet, and weight reloading is required as the constrained 100 mm^2 is insufficient.

IV. Evaluation Set-up and Results

We evaluate the overhead of reconfigurable design with different network models by modifying a widely used DNN+NeuroSim framework [15]. For the simulation set-up, 32 nm node is chosen considering the availability of FeFET in industrial 28nm platform today [4]. FeFET device parameters include on-state resistance=240kΩ, on/off ratio=100, and 4 bit per cell [4]. Sub-array size is fixed to be 128×128, and 5-bit precision ADC is utilized to maintain inference accuracy with multi-bit memory cells [15]. A PE is comprised by 4×4 subarrays and a tile consists of 3×3 PEs. This is the most common structure for custom design of novel mapping to maximize the memory utilization. The customized conventional mapping tiles have 2×2 PEs with the same size. The chip area is constrained to 100 mm^2. With these settings, the chip is assumed to be pre-fabricated. The selected models for runtime reconfiguration include VGG-8 for CIFAR-10, ResNet-18, GoogLeNet, AlexNet and DenseNet-121 for ImageNet. Relatively high precision with 8-bit weight and 8-bit activation is used for inference to ensure no accuracy loss. A global buffer of 2MB is reserved on-chip. The DRAM is implemented by LPDDR4 with bandwidth of 34GB/s and energy of 1.3pJ/bit [19], considering low-power edge applications. We compare the metrics of chip area, memory utilization, throughput and energy efficiency to explore the overhead of runtime reconfiguration.

According to the benchmark results shown in Fig. 11 and 12, the reconfigurable design mainly has overhead of increased area, lower memory utilization and degraded energy efficiency. For example, the used area of ResNet-18 on reconfigurable chip increased by 20% than the custom one because of additional peripheral circuits and mismatch of CM tile structure. The area overhead is increased to 39% for VGG-8 as more CM tiles are required to map this model. The needed area on GoogleNet, AlexNet and DenseNet-121 exceeds 100 mm^2 thus requiring weight reloading. For the overhead of memory utilization, one contributor is less-optimized CM tile structure compared with custom design, and the other is the wasted memory for tiny models mapped on a fixed large chip or for large models during weight reloading cycles. Overall the waste of utilization is relieved as the models evolve to deeper networks, i.e. from 78% of VGG-8 to 27% of DenseNet-121. The throughput and energy efficiency are sacrificed slightly in general. For large models like GoogleNet, AlexNet and DenseNet, the throughput even gets improved because of a shorter data transmission distance from greatly reduced chip area. Nevertheless, the energy efficiency of these large models drops by around 30%~40% due to frequent access to DRAM during weight reloading.

Fig. 13 shows the runtime reconfiguration cost in terms of the latency and energy for switching to a new network model, which are nearly proportional to the number of parameters of the model. Such cost is mainly associated with the programming of the new weights into the FeFET arrays. The average runtime reconfiguration speed and energy cost are around 0.041ms/MB and 0.015mJ/MB, which is much more efficient than FPGA 1.25~2.5ms/MB (reported for UltraScale+ and Virtex-7 [20]) and ~5mJ/MB (reported for Virtex-5 [21]). Table I shows performance comparison and trade-offs between different platforms. Our throughput and energy efficiency is better than a representative ASIC design [22]. Although it may not be as fast or efficient as state-of-the-art ASIC design which considers the model sparsity [23], the support of different CNN models and lower power are our strengths. Compared to a representative FPGA [24], our design outperforms by 6× throughput and 1400× energy efficiency. Compared with state-of-the-art GPU NVidia A100 fabricated on the latest 7nm CMOS, our design shows 2500× lower power consumption and 2.7× energy efficiency. Although GPU shows 900× higher throughput, its ultra-high power makes it prohibitive to be deployed to the edge devices.

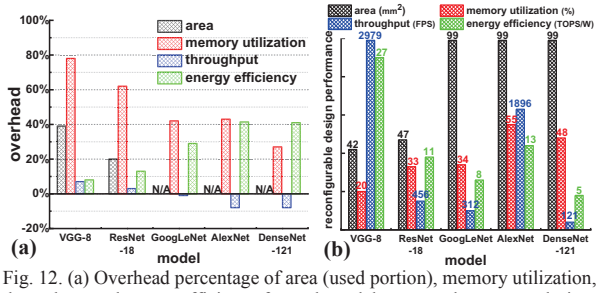

Fig. 12. (a) Overhead percentage of area (used portion), memory utilization, throughput and energy efficiency for each model compared to custom design. (b) Absolute values of area (used portion), memory utilization, throughput and energy efficiency of each model with reconfigurable design.

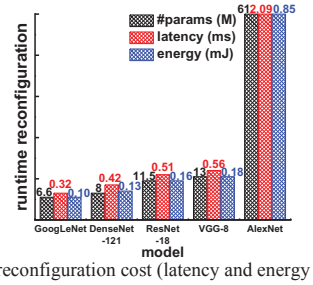

Fig. 13. Runtime reconfiguration cost (latency and energy for loading a new model) vs. # parameters of each model.

TABLE I. PERFORMANCE COMPARISON TABLE

	UNPU [22]	**LNPU [23]**	**[24]**	**[25]**	**This work**
Platform	65nm ASIC	65nm ASIC	Xilinx ZCU102	NVidia Tesla A100	32nm FeFET
Network	AlexNet	VGG-16	AlexNet	---	AlexNet
Precision	1b-16b	8b/16b	16b	16b-64b	8b
Throughput (GOP/S)	691.2 (INT8) 345.6 (INT16)	9285 (FP8) 5725 (FP16)	223.4	1248000 (INT8)	1358.7
Energy efficiency (GOPS/W)	2304 (INT8) 1152 (INT16)	25300 (FP8) 15600 (FP16)	9.5	4992	13310.4
Power (W)	0.3	0.37	23.6	250	0.10

V. CONCLUSION

In this work, a reconfigurable CIM-based accelerator for CNN inference is investigated from the aspects of tile-level reconfigurable mapping, PE-level reconfigurable data transmission and accumulation separately on input and output side. This design supports runtime reconfiguration, which is compatible to majority of CNN family with a fixed predefined chip instance even when the network size is larger than the capacity of on-chip memory. The overhead of this reconfigurable design depends on specific model, as the area can be increased by 20%~40% for tiny models but will be constrained for large models. The unused memory varies from 45% to 80% in our evaluation. Throughput and energy efficiency are normally decreased less than 10%. Especially for large models, throughput can even get improved, while weight reloading from DRAM could cause 30%~40% degradation of energy efficiency to them, which seems inevitable. The runtime reconfiguration is proved to be much more efficient than FPGA. Our design has additional reconfigurability and lower power than ASIC CMOS design; much higher throughput and energy efficiency than FPGA implementation; and ultra-low power compared with GPU. The design methodologies presented in this paper could be applied to chip instances with other predefined on-chip resources, or SRAM and other eNVM based CIM designs.

ACKNOWLEDGMENT

This work is supported in part by NSF-CCF-1903951, and ASCENT, one of the SRC/DARPA JUMP Centers.

REFERENCES

[1] L. Deng, et al. “Model compression and hardware acceleration for neural networks: A comprehensive survey”, *Proc. IEEE*, 108(4), 485-532, 2020.

[2] C.-X. Xue, et al. “A 22nm 2Mb ReRAM compute-in-memory macro with 121-28TOPS/W for multibit MAC computing for tiny AI edge devices,” *IEEE International Solid-State Circuits Conference (ISSCC)*, 2020.

[3] G. W. Burr, et al. “Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses) using phase-change memory as the synaptic weight element,” *IEEE Transactions on Electron Devices*, 62(11), 3498–3507, 2015.

[4] P. Wang, et al. “Drain-erase scheme in ferroelectric field-effect transistor—part I: device characterization,” *IEEE Transactions on Electron Devices*, vol. 67, no. 3, pp. 955-961, 2020.

[5] A. Krizhevsky, et al. “ImageNet classification with deep convolutional neural networks,” *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[6] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *International Conference for Learning Representations (ICLR),* 2015.

[7] C. Szegedy, et al. “Going deeper with convolutions,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2015.

[8] K. He, et al. “Deep residual learning for image recognition,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2016.

[9] G. Huang, et al. “Densely connected convolutional networks,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2017.

[10] M. Sandler, et al. “MobileNet v2: Inverted residuals and linear bottlenecks,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2018.

[11] M. Tan, et al. “EfficientNet: Rethinking model scaling for convolutional neural networks,” *International Conference on Machine Learning (ICML), 2019.*

[12] P. Chi, et al. “PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory,” *ACM/IEEE International Symposium on Computer Architecture (ISCA),* 2016.

[13] A. Shafiee, et al. “ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” *ACM/IEEE International Symposium on Computer Architecture (ISCA),* 2016.

[14] L. Song, et al. “Pipelayer: A pipelined ReRAM-based accelerator for deep learning,” *IEEE International Symposium on High Performance Computer Architecture (HPCA),* 2017.

[15] X. Peng, et al. “DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies,” *IEEE International Electron Devices Meeting (IEDM),* 2019. Available at https://github.com/neurosim/DNN_NeuroSim_V1.2

[16] T. Gokmen, et al. “Training deep convolutional neural networks with resistive cross-point devices,” *Frontiers in Neuroscience*, 11, 538, 2017.

[17] X. Peng, et al. “Optimizing weight mapping and data flow for convolutional neural networks on processing-in-memory architectures,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(4), 1333-1343, 2019.

[18] A. Lu, et al. “Benchmark of the compute-in-memory based DNN accelerator with area constraint,” *IEEE Transactions on VLSI Systems*, *28*(9), 1945-1952, 2020.

[19] D. Skinner, “LPDDR4 moves mobile,” *JEDEC Mobile Forum 2013.*

[20] M. Damschen, et al. “WCET guarantees for opportunistic runtime reconfiguration. ” *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

[21] J. Olivito, et al. “Analysis of the reconfiguration latency and energy overheads for a Xilinx Virtex-5 field-programmable gate array,” *IET Computers & Digital Techniques,* 12(4), 150-157, 2018.

[22] J. Lee, et al. "UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision." *IEEE International Solid-State Circuits Conference (ISSCC)*, 2018.

[23] J. Lee, et al. "LNPU: A 25.3 TFLOPS/W sparse deep-neural-network learning processor with fine-grained mixed precision of FP8-FP16." *IEEE International Solid-State Circuits Conference (ISSCC)*, 2019.

[24] L. Lu, et al. "An efficient hardware accelerator for sparse convolutional neural networks on FPGAs." *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM),* 2019.

[25] NVIDIA, "NVIDIA A100 tensor core GPU," 2020.