A Convolution Neural Network Accelerator Design with Weight Mapping and Pipeline Optimization

Lixia Han, Peng Huang*, Zheng Zhou, Yiyang Chen, Xiaoyan Liu, Jinfeng Kang
School of Integrated Circuits, Peking University
Email: phwang@pku.edu.cn

2023 60th ACM/IEEE Design Automation Conference (DAC) | 979-8-3503-2348-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/DAC56929.2023.10247977

ABSTRACT

The pipeline is an efficient solution to boost performance in non-volatile memory based computing in memory (nvCIM) convolution neural network (CNN) accelerators. However, the previous works seldom focus on pipeline optimization from the perspective of the whole system, especially overlooking the effect of buffer access. In this work, we propose a high-performance NVM-based CNN accelerator with a balanced pipeline design, which takes account of both the macro computing and the buffer access. At the operator level, a matrix-based weight mapping method is proposed to reduce buffer access delay. At the macro level, decoupled access and execution design is introduced to shorten the single-layer latency. At the system level, a hybrid inter/intra-tile design is presented to balance the overall latency across CNN layers. With the collaboration among three methods, we construct a well-balanced pipeline for the nvCIM accelerator at a smaller hardware cost. Experiments show that our pipeline design can achieve 3.7x, 7.5x, and 3.5x throughput improvement for recognition of ImageNet with ResNet18, VGG19, and ResNet34 models, respectively.

1. INTRODUCTION

In-memory computing (CIM) benefitting from in situ computing and high parallelism computing shows great potential to improve energy efficiency and throughput in data-centric application scenarios[1-5]. Various emerging non-volatile memory (NVM), such as RRAM[6-8], PCM[9, 10], and FeFET[11, 12] are promising candidates owing to their high storage density and low computing consumption. With the NVM crossbar structure, vector-matrix multiplication (VMM) can be efficiently performed according to Ohm's Law and Kirchhoff's Law. Lots of NVM-based CIM (nvCIM) macros have been proposed with >10TOPS/W energy efficiency and >100GOPS throughput[7-10]. Efficient VMM computing ability in nvCIM macros agrees with the requirements of the inference of convolution neural network (CNN) algorithms, which account for more than 90% of total computation[13]. Therefore, the nvCIM accelerator is widely regarded as a promising route in CNN hardware accelerators. The nvCIM architectures for CNN algorithms have been proposed in several works, e.g. ISAAC[14], PRIME[15], which achieve more than hundreds of times improvement in both energy efficiency and computation throughput. The fully hardware-implemented nvCIM accelerators [16, 17] for CNN algorithms have also been verified in principle and exhibited minimal accuracy losses and high energy efficiency.

The critical characteristic of nvCIM accelerators compared with other hardware platforms is the weight pre-storage. For CNN inference tasks, all weights of CNN model have been programmed into the NVM conductance states in advance. Therefore, improving the time utilization of CIM macro is a preferred solution to improve the system throughput of nvCIM accelerators. However, in deep CNNs, the output feature maps (OFMs) of the current layer are also the input feature maps (IFMs) of the next layer. If the computation of the next layer starts until the whole OFMs from the current layer is completed, most of the hardware resources are idle for a long period of time. It would result in unnecessary waste of hardware resources and further reduce the throughput of nvCIM accelerators.

Previous works[14, 18, 19] have proposed pipeline structures to minimize the idle time of hardware resources in nvCIM accelerators. The CIM macros are pipelined by computing parallelly across layers instead of layer by layer, that is, once the IFMs are sufficient to compute convolution, the computation is triggered. But, in actuality, this raw pipeline is often stuck by a bottleneck layer due to inconsistent parameters across CNN layers. To further decrease the idle time, the intra-tile pipeline is proposed to speed up the bottleneck layer through duplicating weights with more hardware. However, the buffer access of IFMs, OFMs, and partial sums, both load and write back, are miss discussed.

Taking the buffer access delay into account, the previous pipeline design focusing on CIM macros would suffer from unbalanced problems again. The reasons include: 1) the buffer access amount is huge, e.g., there are 14M IFMs/OFMs and 162M partial sums in VGG19 model when CIM macro size is 16Kbit; 2) the buffer access amount varies in different layers due to network structure, causing unbalanced buffer access delay across layers; 3) repeated IFMs access occurred by convolutional operation further magnified the unbalanced buffer access delay.

In this work, from the perspective of the whole system, we propose a high-performance NVM-based CNN accelerator with a balanced pipeline, which both takes account of the CIM macro computing delay and buffer access delay.

DAC'23 July 9-13, 2023, San Francisco, CA, USA
ACM ISBM 979-8-3503-2348-1/23/$31.00 ©2023 IEEE

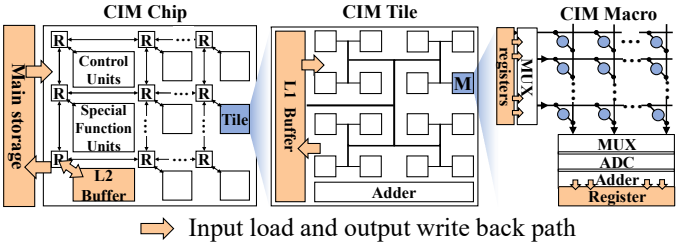

Fig. 1. The diagram of NVM-based CIM hierarchy architecture.

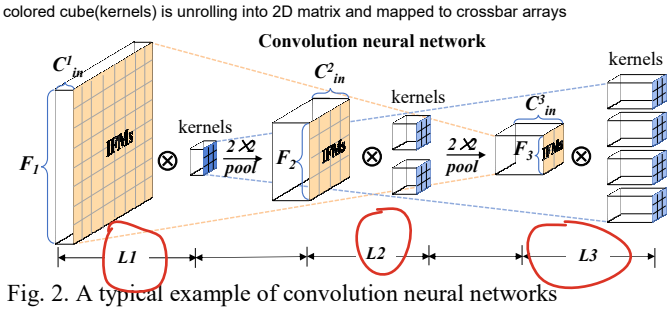

Fig. 2. A typical example of convolution neural networks

The contributions of this work can be listed as follows:

- At the operator level, a matrix-based weight mapping method is introduced to reduce the buffer access delay of a single convolution operation by reusing data locally.
- At the macro level, a decoupled access and execution design is proposed to shorten single-layer delay by overlapping the access latency and computation latency.
- At the system level, a hybrid inter/intra-tile pipeline method is designed to balance the overall latency across CNN layers by assigning computing tasks to multiple tiles.
- 3.7x, 7.5x, and 3.5x throughput boosts are achieved compared with baseline (intra-tile pipeline) for ImageNet recognition with ResNet18, VGG19, and ResNet34 models.

2. BACKGROUND

2.1 NVM-based CIM architecture

Fig. 1 shows the diagram of the NVM-based CIM hierarchy architecture. CIM chip mainly includes multiple tiles, a control unit, special function units, and L2 buffer, which are connected by a mesh network on chip. Each CIM tile generally consists of multiple macros, L1 buffer, and adders. The routing among them is based on the H-tree structure. Each macro mainly contains NVM crossbar, input/output registers, row/column multiplexers, analog-digital converters, and adders.

During CNN inference, the weights are stored in conductance states in advance. IFMs are successively loaded from the L2 buffer, L1 buffer, input registers and then sent to each tile, macro, NVM crossbar. After the VMM computation between IFMs and weights, the output current is converted to digital signal by peripheral ADC circuits and stored in the output registers. The output partial sums from various macros are sent to adders and then written back L1 buffer. Finally, the OFMs need to be activated and pooled in special function units and written back to L2 buffer.

2.2 CNN implementation in nvCIM accelerators

Convolution layers in CNN extract feature through sliding shared convolution kernels on IFMs. Along CNN layers, convolution kernels with more channels are used to extract the complex deep features and pooling layers are adopted to reduce the IFMs dimensions, as shown in Fig. 2. When implementing CNN models with nvCIM accelerators, the shallow layer generally with fewer weight parameters needs to compute frequently ($F_1 \times F_1$ times), while the deep layer with more weight parameters requires to compute sparsely ($F_3 \times F_3$ times). Therefore, most of the hardware resources of deeper layers are idle for a long time, causing a decline in system throughput.

2.3 Previous pipelines

Pipeline is an efficient solution to utilize hardware resources and reduce overall latency. Previous pipelines mainly include the raw pipeline and the intra-tile pipeline, as shown in Fig.3. The raw pipeline design performs parallel computation across CNN layers. However, due to the IFMs size of shallower layers being much bigger than that of deeper layers after several pooling operations, the raw pipeline is stuck by shallower layers, which spend more time for both buffer access and macro computation, as shown in Fig.3(a).

To further optimize overall latency, the intra-tile pipeline speeds up the computation of bottleneck layers through duplicating weights of bottleneck layers, as shown in Fig. 3(b). However, duplicating weights and mapping them to the same tile only reduces the computation delay, not IFMs and OFMs access delay. Taking buffer access delay into account, intra-tile pipeline design still suffers from unbalanced challenges.

2.4 Buffer access analysis of CNN accelerators

The buffer access in nvCIM accelerators shows the following three characteristics: First, each IFM point needs to be convoluted multiple times. Thus, IFMs need to be accessed repeatedly, increasing the buffer access amount.

Second, the buffer access delay is limited by bandwidth. The serial access and execution design make the buffer delay occupies a high proportion of the total latency, resulting in the CIM macro being idle for a longer period of time.

Third, the buffer access delay varies across layers due to different IFMs sizes, input channels, and kernel sizes. Therefore, the intra-tile pipeline may be stuck by the bottleneck layer with the longest buffer access delay.

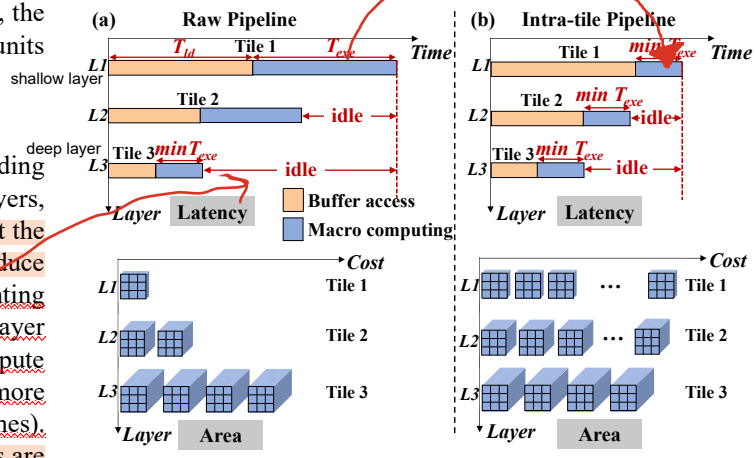

Fig. 3. The previous pipeline designs of CNN accelerator, (a) raw pipeline, (b) intra-tile pipeline.

cok guzel bir noktayi yakalamis ve onerisine bakalim-->

Weight Mapping to the Crossbar:

Each colored cube (kernel) is unrolled into a 2D matrix and mapped to the crossbar arrays in the CIM hardware. For example, a k x k x Cin kernel becomes a k x k x Cin row vector in the crossbar.

The colored cubes (representing the weights of the convolution kernels) are mapped to the crossbar arrays in the CIM hardware during the initialization or setup phase of the convolution operation for a specific layer. This process occurs before the input feature maps (IFMs) start interacting with the crossbar for computation. Here's a breakdown of when and how this happens:

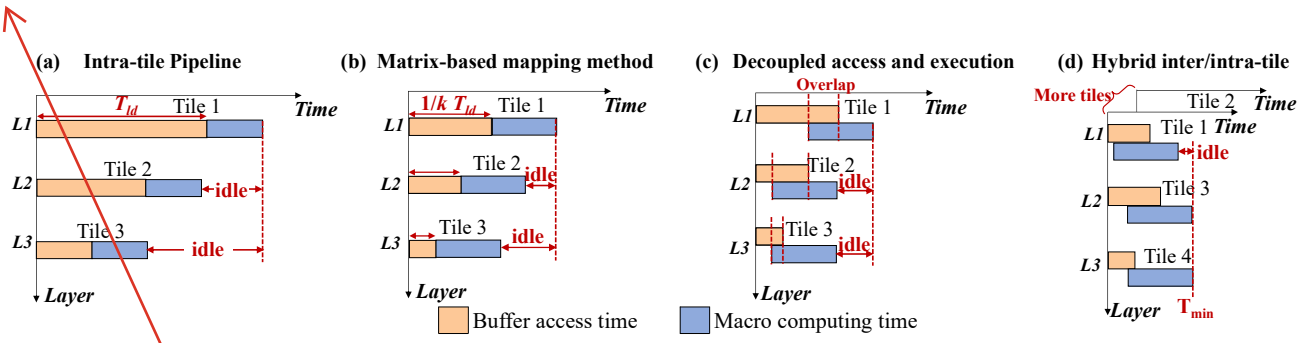

Fig. 4. Overview of pipeline optimization design (a) baseline (intra-tile pipeline), (b) matrix-based mapping method, (c) decoupled access and execution, (d) hybrid inter/intra-tile design.

3. BALANCED PIPELINE DESIGN

3.1 Overview

Intra-tile pipeline can effectively improve the throughput of CNN accelerators, but it suffers from unbalanced problems when taking buffer access delay into account. On the basis of the intra-tile pipeline, we propose a high-performance NVM-based CNN accelerator with a balanced pipeline design, which takes account of both CIM macro computing latency and buffer access latency. In terms of three buffer access characteristics in Section II, we propose three optimization methods to construct a well-balanced pipeline, as shown in Fig.4. First, a matrix-based CNN mapping method is proposed to release the repeated access of IFMs. Second, decoupled access and execution is introduced to reduce the single-layer latency by overlapping access and computation in time. Third, a hybrid inter/intra-tile pipeline design is designed to balance the overall latency across CNN layers by assigning computing tasks of the bottleneck layer to multiple tiles.

3.2 Matrix-based mapping method

In convolution layers, multiple 3D kernels (C_{out}, C_{in}, k, k) extract features by sliding on the 3D input feature map (C_{in}, F_i, F_i) to obtain 3D output feature map (C_{out}, F_o, F_o). When mapping CNN models on CIM chips, the 2D weights of the fully connected layer can be directly mapped to the crossbar, but multiple 3D kernels of the convolutional layer need to be unrolled to the 2D matrix, and the 3D IFM windows also need to be converted into 1D vector, thus convolution can be implemented naturally with NVM crossbar.

The native weight mapping method unrolls each 3D kernel into a ($k \times k \times C_{in}$) column and multiple 3D kernels are transformed into a weight matrix (C_{out}, $k \times k \times C_{in}$) [20]. At each sliding cycle, $k \times k \times C_{in}$ size IFM data is loaded from buffers, as shown in Fig. 5(a). To reuse the IFM data rather than repeatedly accessed from buffers, a matrix-based weight mapping method is proposed. Each 3D kernel is unrolled into the partial matrix (k, $k \times C_{in}$) and multiple 3D kernels are transformed into a whole matrix ($k \times C_{out}$, $k \times C_{in}$). At each sliding cycle, $k \times C_{in}$ size IFM data is loaded from buffers and input to the crossbar, as shown in Fig. 5(b). Each IFM point in the sliding cycle is multiplied by $k \times C_{out}$ size weights rather than C_{out} size weights in [20]. Therefore, the matrix-based CNN mapping method reduces the loaded IFMs amount per convolution operation to $1/k$ times that of the native mapping method by reusing the loaded IFMs locally.

Blue, green and purple cubes represents the weights of the convolution kernels.

These cubes extracts specific futures from IFM. Each kernel focus on extracting unique feature.

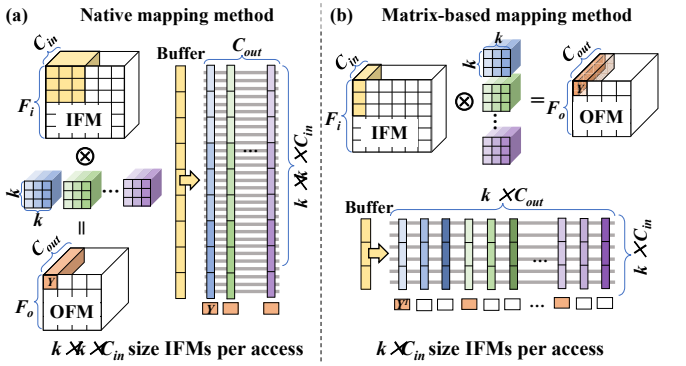

Fig. 5. Convolution neural network mapping method, (a) native mapping method, (b) matrix-based mapping method.

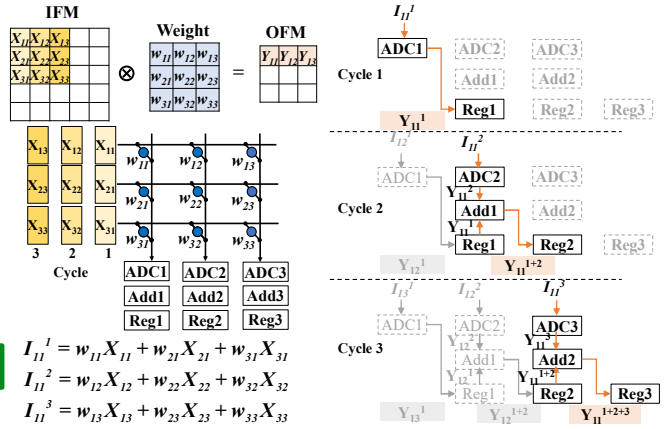

Fig. 6. The implementation of the matrix-based mapping method.

Fig. 6 shows the hardware implementation for the matrix-based mapping method. To clearly illustrate the implementation of convolution, we assume kernel size is 3×3 and only show one output value (Y_{11}) in one output channel. In fact, in each sliding cycle, $k \times C_{out}$ size OFMs are obtained simultaneously. At sliding cycle 1, the IFMs (X_{11}, X_{21}, X_{31}) are loaded from the buffer and input into the NVM crossbar and then perform multiplication and accumulation (MAC) with weights (w_{11}, w_{21}, w_{31}). The output current I_{11}^1 is converted by ADC1 and digital output Y_{11}^1 is stored in Reg1. At sliding cycle 2, the IFMs (X_{12}, X_{22}, X_{32}) are loaded and performed MAC with weights (w_{12}, w_{22}, w_{32}). Y_{11}^2 converted by ADC 2 and Y_{11}^1 stored in Reg1 are added and the partial sum Y_{11}^{1+2} is stored in Reg2. Similarly, the final output Y_{11} is obtained by adding Y_{11}^3 and Y_{11}^{1+2} and then stored in Reg3. Generally, each output result can be obtained by summing up adjacent k columns MAC results of k consecutive times.

The shallow layer is not necessarily "simpler" than the deep layer in terms of the number of operations, but it has smaller filters and more frequent computations over the image.

The deep layer, with its larger filters and more channels, ends up doing fewer operations in terms of number of positions but requires more computational resources per position. It processes a larger number of parameters but on a smaller feature map.

how it is possible to have more frequent computations with smaller filter

because I am looking for every detail on the image is not it?

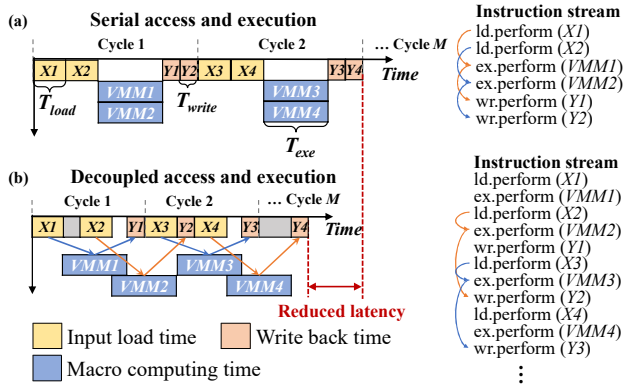

Fig. 7. Time sequence of (a) serial access and execution, (b) decoupled access and execution

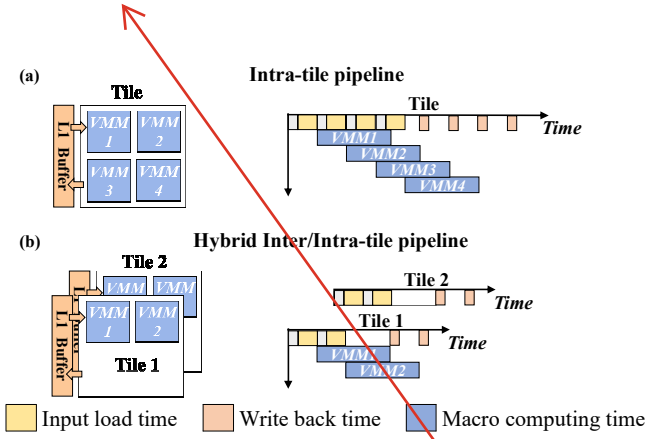

Fig. 8. Weight mapping and time sequence of (a)intra-tile pipeline, (b) hybrid inter/intra-tile pipeline.

aslında shallow da pencere boyutu daha küçük bu sebeple resim üzerinde her detaya bakıyorum ve çok daha fazla computation gerekiyor bu sebeple boş zaman kalmıyor
ama deep layer'da pencere boyutun daha büyük bu sebeple daha az computation oluyor ve idle zaman kalıyor

3.3 Decoupled access and execution design

Because of limited buffer bandwidth and control complexity, buffers only perform read or write operations at the same time. In nvCIM accelerators, the time sequence of serial access and execution is firstly loading input data from the buffer to CIM macro, then computing convolution, and finally writing the output back into the buffer. As shown in Fig.7(a), although CIM macros can compute convolution *VMM1* and *VMM2* parallelly, the input data *X1* and *X2* are loaded and the output data *Y1* and *Y2* are written back serially. The total latency of N_p CIM macros for M convolution cycles is shown in formula (1).

$$T_{total} = ((T_{load} + T_{write}) \times N_p + T_{exe}) \times M \quad (1)$$

T_{exe} is the macro computing delay, T_{load} is the input data load delay and T_{write} is the output data write-back delay

We introduce decoupled access and execution design into nvCIM accelerators, which reduces the single-layer latency by allowing access and computation to overlap in time, as shown in Fig. 7(b). We analyze the reduced latency when the macro computation delay is longer than or shorter than the total buffer access delay, respectively. If the computation delay is longer than the total buffer access delay, the computation delay would take up most of the total latency. The total latency of N_p CIM macros for M convolution cycles can be calculated as formula (2):

$$T_{total} = (T_{exe} + T_{load} + T_{write}) \times M + (T_{load} + T_{write}) \times (N_p - 1) \quad (2)$$

Decoupled access and execution design reduces $(M\text{-}1)\times(N_p\text{-}1)\times(T_{load}+T_{write})$ size latency. If the computation delay is shorter than the total buffer access delay, the buffer access delay would dominate the total latency. The total latency of N_p CIM macros for M convolution cycles can be calculated as formula (3):

$$T_{total} = (T_{load} + T_{write}) \times (N_p \times M + N_p - 1) \quad (3)$$

$T_{exe}\times M$ - $(N_p\text{-}1)\times(T_{load} + T_{write})$ size latency is reduced with decoupled access and execution design.

3.4 Hybrid inter/intra-tile design

Intra-tile pipeline design reduces the overall latency through duplicating weights of bottleneck layers and mapping them in the same tile. Although multiple duplicated weights can be computed at the same time, the IFMs and OFMs need to be loaded from and written back to buffers sequentially. Therefore, the intra-tile pipeline still suffers from unbalanced challenges from nonnegligible buffer access delay.

A solution to the above problem is mapping weight duplications into independent tiles. As described in Section II, the bottleneck layer of CNN pipeline generally is the shallow layer, and the weight parameters in the shallow layer are generally much smaller than the CIM tile size. Therefore, mapping small weights into large tiles reduces latency at the expense of large area overhead.

A hybrid inter/intra-tile design combining throughput benefit and area cost is proposed to balance the overall latency among different CNN layers. As shown in Fig. 8, the four weight duplications are mapped into two tiles, even though a single tile can accommodate them. Assigning weight duplications of bottleneck layers over multiple tiles reduces the buffer access amount of a single tile, which in turn reduces the latency of bottleneck layers. There is a tradeoff between weight duplications in the same tile and tile numbers. More weight duplications in a single tile would increase the tile latency and more tile numbers would increase area.

3.5 Well-balanced pipeline scheme

Algorithm 1 provides a well-balanced pipeline optimization scheme. The optimization goal is obtaining greater throughput gains at a smaller area cost. We evaluate the optimization index P_{LA} (the product between system latency T_{min} and area costs $\sum N_i^l$) when taking the delay of any CNN layer (L) as the overall balanced latency (T_{min}). If L layer exhibits the minimum P_{LA}, then the delay of L layer is set to the balanced latency of the optimized pipeline.

In terms of the delay and area evaluation of each layer, we first unroll convolution kernels into a weight matrix [H_l, W_l] according to the matrix-based mapping method and then compute the latency (T_l) without pipeline design. If T_l is shorter than desired balanced latency (T_{min}), we just evaluate its area cost (N_i^l). If T_l is longer than desired balanced latency (T_{min}), the required total weight duplication numbers (N_c^2) in pipeline and the maximum duplication numbers (N_t) single tile can hold are evaluated. The initial duplication numbers N_p^l in the same tile are determined by the minimum value between N_c^2 and N_t. According to T_{exe} and T_{ld}^l, we re-compute the latency T_l with decoupled access and execution design. We decrease N_p^l until T_l is no longer than T_{min}. Finally, area cost (N_i^l) is evaluated.

Algorithm1 to optimize pipeline scheme

Input: 1. DNN model C_{in}, C_{out}, k, k, F

2.Tile size S_{tile}, buffer buswidth BW, macro computing delay T_{exe}

for L = *layer* -- **do**

  $[H_L, W_L] = [k_L \times C_{in}^{\,L},\ k_L \times C_{out}^{\,L}]$

  $T_{min} = (T_{exe} + H_L / BW + C_{out}^{\,L} / BW) \times F_L \times F_L$

  for l = *layer*-- **do**

    $[H_l, W_l] = [k_l \times C_{in}^{\,l},\ k_l \times C_{out}^{\,l}]$

    $T_{ld}^{\,l} = H_l / BW + C_{out}^{\,l} / BW$

    $T_l = (T_{exe} + T_{ld}^{\,l}) \times F_l \times F_l$

    If $T_l \le T_{min}$ **do**

      $N_i^l =$ ***ceil*** $((H_l \times W_l) / S_{tile})$

    else do

      $N_c =$ ***floor*** (F_l / F_L)

      $N_t =$ ***floor*** $(S_{tile} / (H_l \times W_l))$

      $N_p^{\,l} = N_t > 0$? ***min***$(N_c^2, N_t) : 1$

      while $N_p^{\,l} > 1$ **do**

        if $T_{exe} > ((N_p^{\,l} - 1) \times T_{ld}^{\,l})$ **do**

          $T_l = (T_{exe} + T_{ld}^{\,l}) \times F_l \times F_l / N_c^2 + (N_p^{\,l} - 1) \times T_{ld}^{\,l}$

        else do

          $T_l = T_{ld}^{\,l} \times F_l \times F_l / N_c^2 \times N_p^{\,l} + (N_p^{\,l} - 1) \times T_{ld}^{\,l}$

        if $T_l > T_{min}$

          $N_p^{\,l}$ --

        else do

          Break

      $N_i^l =$ ***ceil*** $(N_c^2 / N_p^{\,l}) \times$ ***ceil*** $((H_l \times W_l) / S_{tile})$

  $P_{LA}[L] = T_{min} \times \sum N_i^l$

$P_{opt} = \min(P_{LA})$

4. EVALUATION

4.1 Experiment setup

1) Networks and dataset

We apply the intra-tile pipeline and optimized pipeline to the implementation of NVM-based CIM acceleration systems for ResNet18, VGG19, and ResNet34 models. The recognition dataset is 224×224 clipped ImageNet. All weights and activations are quantized to 8bit.

2) CIM parameters and performance

All components of the CIM accelerator are evaluated under 32nm technology node. We use CACTI[21] to evaluate energy, delay, and area of L2 buffer, L1 buffer, and Register, as listed in Tab. I. In CIM macro, the NVM subarray and ADC dominate the energy and area[14, 22]. For ADC energy and area, we use data from the[23], which reported a 35mW 8bit ADC with 0.003mm^2 active area in 32nm technology node. For NVM subarray energy and area, we use data from [16], in which the RRAM cell shows 32 independent conductance states under 0.2V read voltage and 10ns pulse width. We set that the NVM subarray size is 128×128 and each tile consists of 64 CIM macros[18]. 16 ADCs are time-multiplexed by 128 columns in each CIM macro.

Tab. I. Summary of nvCIM parameters and performance

		Size (KB)	Bandwidth /numbers	Energy (nJ)	Delay (ns)	Area (mm^2)
CIM chip	L2 buffer	256	256bit	0.06/access	0.003/bit	0.7
	Tile	1Mb	/	69/op	/	0.51
Tile	L1 buffer	64	256bit	0.03/access	0.002/bit	0.13
	Macro	16Kb	64	0.13/op	/	0.06
Macro	Sub array		128×128	0.065	10	0.0002
	ADC	8bit	16	0.0005	0.12/op	0.003
	Reg	4KB	128bit	0.002/access	0.001/bit	0.008

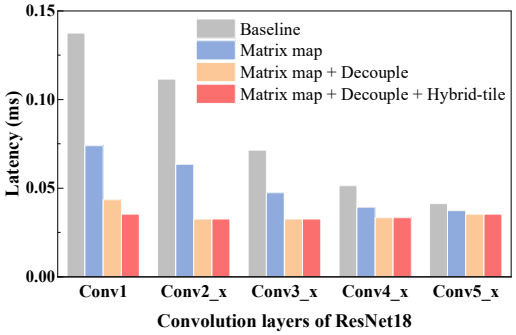

Fig. 9. The latency comparison of ResNet18 among the baseline pipeline, optimized pipeline with matrix map method, decoupled design, and hybrid-tile design.

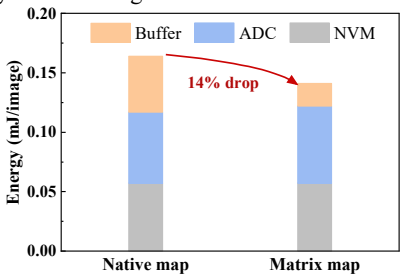

Fig. 10. The energy consumption comparison of ResNet18 between the native map method and the matrix map method.

4.2 Experiment results

We compare the system latency of nvCIM accelerator for ResNet18 model between the intra-tile pipeline (baseline) and optimized pipeline by gradually introducing the matrix-based mapping method, decoupled access and execution, and hybrid inter/intra-tile design. These designs can be used individually to optimize the pipeline, and the collaboration among them makes it possible to build a well-balanced pipeline at a smaller hardware cost.

Fig. 9 shows that the latency decreases along CNN layers with the baseline pipeline. It can be found that the hardware resources of Conv2_x-Conv5_x are idle for a long period of time. The overall latency depends on the Conv1 layer with the highest latency.

To construct a well-balanced pipeline, we first adopt the matrix-based mapping method to reduce the amount of buffer access per convolution operation. Fig. 9 shows the latency of all CNN layers is reduced. Meanwhile, less amount of buffer access can also save buffer access energy. Fig. 10 shows matrix-based mapping method can save 14% energy consumption for the ImageNet recognition task.

Then, decoupled access and execution design is applied to the pipeline to shorten the single-layer latency by allowing the access and computation to overlap in time. Fig. 9 shows that the latency of Conv1-Conv5_x continues to be reduced. We can observe that the latency gap from Conv2 to Conv5_x is subtle, but the system pipeline is still stuck on the Conv1 layer.

Finally, we assign computation tasks of the Conv1 layer to multiple independent tiles to speed up Conv1 layer. Fig. 9 shows that a well-balanced pipeline has been constructed with comprehensive optimization from matrix-based mapping design, decoupled access and execution design, and hybrid inter/intra-tile design.

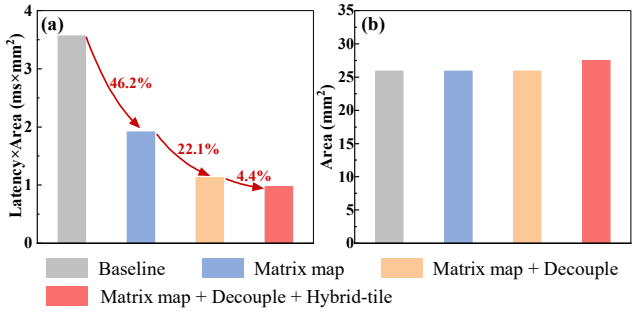

Fig. 11. The comparison results of ResNet18 among baseline pipeline and optimized pipeline, (b) latency×area, (a) area overhead

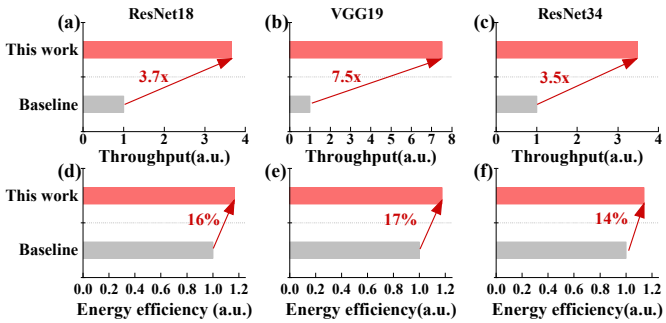

Fig.12. The throughput and energy efficiency for baseline pipeline and optimized pipeline, (a)(d) ResNet18, (b)(e) VGG19, (c)(f) ResNet34.

Fig. 11(a) shows the index of latency×area for the optimized pipeline, which is decreased by 46.2%, 22.1% and 4.4% for matrix-based mapping method, decoupled access and execution design, and hybrid inter/intra-tile design, respectively. Fig. 11(b) shows the area overhead for the optimized pipeline. No additional area is occupied for matrix-based mapping design and decoupled access and execution design. The area is slightly increased when using hybrid inter/intra-tile design to balance the overall latency. Using Algorithm 1, the additional area has been optimized as much as possible.

We apply the baseline pipeline and optimized pipeline to the nvCIM accelerators for ImageNet recognition with ResNet18, VGG19, and ResNet34 models. Fig. 12 shows that optimized pipeline achieves 3.7x, 7.5x and 3.5x throughput boost and 16%, 17% and 14% energy efficiency improvement for ResNet18, VGG19, and ResNet34 models, respectively.

5. CONCLUSION

In this work, we present an NVM-based CNN accelerator design with the insight of the whole system. Taking both CIM macro computing delay and buffer access delay into account, we optimize the pipeline from three levels. At the operator level, a matrix-based convolution mapping method is proposed to reduce both buffer access delay and buffer access energy consumption by reusing data locally. At the macro level, a decoupled access and execution design is introduced to shorten the delay of single layer by overlapping the access latency and computation latency. At the system level, a hybrid inter/intra-tile scheme is designed to balance the overall latency across CNN layers by assigning computing tasks into multiple tiles. Experiments show that the optimized pipeline can achieve 3.7x, 7.5x, and 3.5x throughput boost for ImageNet recognition with ResNet18, VGG19, and ResNet34 models, respectively.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under Grant 62022006, Grant 92064001, and Grant 62104007 and in part by the 111 Project (B18001).

REFERENCES

[1] K. Roy *et al.*, "In-memory computing in emerging memory technologies for machine learning: an overview," in *DAC*, 2020: IEEE, pp. 1-6.
[2] W. Zhang *et al.*, "Neuro-inspired computing chips," *Nature electronics,* vol. 3, no. 7, pp. 371-382, 2020.
[3] A. Sebastian *et al.*, "Memory devices and applications for in-memory computing," *Nat. Nanotechnol.,* vol. 15, no. 7, pp. 529-544, 2020.
[4] M. Zhou *et al.*, "Mat: Processing in-memory acceleration for long-sequence attention," in *DAC*, 2021: IEEE, pp. 25-30.
[5] M. Li *et al.*, "iMARS: an in-memory-computing architecture for recommendation systems," in *DAC*, 2022, pp. 463-468.
[6] J. Yue *et al.*, "AERIS: Area/Energy-efficient 1T2R ReRAM based processing-in-memory neural network system-on-a-chip," in *ASP-DAC*, 2019, pp. 146-151.
[7] J.-M. Hung *et al.*, "An 8-Mb DC-Current-Free Binary-to-8b Precision ReRAM Nonvolatile Computing-in-Memory Macro using Time-Space-Readout with 1286.4-21.6 TOPS/W for Edge-AI Devices," in *ISSCC*, 2022, vol. 65: IEEE, pp. 1-3.
[8] C.-X. Xue *et al.*, "A 22nm 4Mb 8b-precision ReRAM computing-in-memory macro with 11.91 to 195.7 TOPS/W for tiny AI edge devices," in *ISSCC*, 2021, vol. 64: IEEE, pp. 245-247.
[9] R. Khaddam-Aljameh *et al.*, "HERMES Core–A 14nm CMOS and PCM-based In-Memory Compute Core using an array of 300ps/LSB Linearized CCO-based ADCs and local digital processing," in *VLSI*, 2021: IEEE, pp. 1-2.
[10] W.-S. Khwa *et al.*, "A 40-nm, 2M-Cell, 8b-Precision, Hybrid SLC-MLC PCM Computing-in-Memory Macro with 20.5-65.0 TOPS/W for Tiny-Al Edge Devices," in *ISSCC*, 2022, vol. 65: IEEE, pp. 1-3.
[11] H. Amrouch *et al.*, "Cross-layer Design for Computing-in-Memory: From Devices, Circuits, to Architectures and Applications," in *ASP-DAC*, 2021, pp. 132-139.
[12] Y. Long *et al.*, "A ferroelectric FET based power-efficient architecture for data-intensive computing," in *ICCAD*, 2018, pp. 1-8.
[13] J. Emer, "Design efficient deep learning accelerator: Challenges and opportunities," in *Proc. VLSI Short Course*, 2017, pp. 1-121.
[14] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News,* vol. 44, no. 3, pp. 14-26, 2016.
[15] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News,* vol. 44, no. 3, pp. 27-39, 2016.
[16] P. Yao *et al.*, "Fully hardware-implemented memristor convolutional neural network," *Nature,* vol. 577, no. 7792, pp. 641-646, Jan 2020.
[17] W. Wan *et al.*, "A compute-in-memory chip based on resistive random-access memory," *Nature,* vol. 608, no. 7923, pp. 504-512, 2022.
[18] X. Peng *et al.*, "Optimizing weight mapping and data flow for convolutional neural networks on processing-in-memory architectures," in *TCAS I: Regular Papers,* vol. 67, no. 4, pp. 1333-1343, 2019.
[19] T. Tang *et al.*, "Binary convolutional neural network on RRAM," in *ASP-DAC*, 2017: IEEE, pp. 782-787.
[20] T. Gokmen *et al.*, "Training deep convolutional neural networks with resistive cross-point devices," *Frontiers in neuroscience,* vol. 11, p. 538, 2017.
[21] N. Muralimanohar *et al.*, "CACTI 6.0: A tool to model large caches," *HP laboratories,* vol. 27, p. 28, 2009.
[22] Q. Liu *et al.*, "33.2 A fully integrated analog ReRAM based 78.4 TOPS/W compute-in-memory chip with fully parallel MAC computing," in *ISSCC*, 2020: IEEE, pp. 500-502.
[23] L. Kull *et al.*, "A 35mW8 b 8.8 GS/s SAR ADC with low-power capacitive reference buffers in 32nm Digital SOI CMOS," in *VLSI*, 2013: IEEE, pp. C260-C261.