FDCA: Fine-grained Digital-CIM based CNN Accelerator with Hybrid Quantization and Weight-Stationary Dataflow

Bo Liu[1,2], Qingwen Wei[1], Yang Zhang[1], Xingyu Xu[1], Zihan Zou[1], Xinxiang Huang[1], Xin Si[1,2], Hao Cai[1,2]
[1]School of Integrated Circuits, Southeast University, Nanjing, China
[2]National Center of Technology Innovation for EDA, Nanjing, China
{liubo_cnasic,weiqingwen,zhangyang23,xingyuxu,zouzihan3,xinxianghuang,xinsi,hao.cai}@seu.edu.cn

ABSTRACT

Digital-Compute-in-memory (DCIM) has demonstrated significant energy and area efficiency in convolutional neural network (CNN) accelerators, particularly for high precision applications. However, to mitigate parasitic effects on word and bit lines, most DCIMs employ fine-grained multiply-accumulate operations, which introduces new challenges and opportunities but has not been widely explored. This paper proposes FDCA: a fine-grained digital-CIM based CNN accelerator with hybrid quantization and weight-stationary dataflow, in which the key contributions are :1) a hybrid quantization approach for CNNs leveraging hessian trace and approximation is utilized. This method incorporates the ratio of computation time and storage time into quantization, achieving high efficiency while maintaining accuracy; 2) a Cartesian Genetic Programming based approximate shift and accumulate with error compensation is proposed, where an approximate adder tree is generated to compensate for errors introduced by DCIM; 3) an optimized weight-stationary dataflow is used to improve the utilization of CIM and eliminate dataflow stalls. The experimental results demonstrate that under 28-nm process, when running VGG16 and ResNet50 on CIFAR100, the proposed FDCA achieves 17.1TOPS/W and 18.79TOPS/W with only a slight decrease in accuracy by 0.71% and 0.98%, respectively. Compared to previous works, this work achieves 1.76× and 1.28× better in energy efficiency with less accuracy loss.

KEYWORDS

compute-in-memory-based accelerator, approximate computing, hybrid quantization, convolution neural networks

1 INTRODUCTION

Given the development of convolutional neural networks (CNNs), the limitations of AI devices based on conventional digital architecture have become evident. The computational and memory demands inherent in neural networks pose challenges for digital devices, as the extensive data movement involved leads to inefficiencies. To address this problem, the Compute-In-Memory (CIM) architecture has emerged as a solution by incorporating multiply-accumulate (MAC) directly into the memory unit, thereby enhancing energy efficiency and overcoming the memory wall issue [1].

In recent years, there has been a significant focus on Digital-Compute-in-memory (DCIM) research to optimize energy and area efficiency while maintaining high accuracy [2]. In comparison to Analog-based CIM, DCIM exhibits superior area efficiency, owing to the absence of Analog-to-Digital or Digital-to-Analog converters. Furthermore, Analog-based CIM is susceptible to variations in process, voltage, and temperature, resulting in suboptimal computation and lower precision outcomes [3][20]. Therefore, DCIM presents itself as a promising computing paradigm for CNNs, offering both high energy and area efficiency without compromising accuracy.

However, based on current research, three challenges remain to be solved. Firstly, current DCIM requires loading of weights from off-chip memory, such as DDR or flash, which leaves memory wall unsolved [4]. Besides, limited by parasitic effects on word and bit lines, DCIM mostly adopts fine-grained MAC, performing 1-bit or 2-bit multiplication at once [5][6][7]. This design makes shift and accumulate units necessary for DCIM based accelerators, which causes additional power consumption. Thirdly, due to long-delay of updating weight, weight-stationary (WS) dataflow is adopted in most DCIM based accelerator, which causes a conflict between low utilization of DCIM and requirement for extra memory [1][8][12].

To solve these challenges, we propose a fine-grained DCIM based CNN accelerator with hybrid quantization and WS dataflow. We consider hybrid quantization based on both the hessian trace and the ratio of computation time and storage time (CSR), aiming at maximizing energy efficiency while maintaining the precision of network inference. Besides, Cartesian Genetic Programming (CGP) is utilized to for approximate computing to compensate the error introduced by DCIM. Moreover, a WS dataflow optimized for fine-grained DCIM is proposed to improve the utilization and avoid extra on-chip memory.

The contributions of this paper are summarized as follows:

- We propose a hessian-trace-based hybrid quantization with approximate computing, aiming at maximizing energy efficiency while minimizing the accuracy loss.
- We propose a CGP based approximate shift and accumulate unit with error compensation, in which a revised sign bit extension is utilized and approximate adders are used to trade-off the error induced by CIM.
- We propose a WS dataflow optimized for fine-grained DCIM which improves the utilization for various size of convolution and avoids extra on-chip memory.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3656253

This paper is organized as follows. Section 2 introduces the background and motivation. Section 3 presents details about FDCA. Section 4 gives evaluation results and discussions and Section 5 concludes this paper.

2 BACKGROUND AND MOTIVATION

2.1 Compute-In-Memory Based Accelerator

The left part of Fig. 1 illustrates the basic elements of a DCIM based accelerator. The weights and features of CNNs are stored in DDR and transferred to accelerator by on-chip bus. Upon commencement of the inference, weights are loaded from DDR and stored in the DCIM and updated after traversal of input feature map (IFM). Fine-grained DCIM executes MAC in DCIM-based processing engine array (DCIM-PEA) and the convolution results are obtained through the shift and accumulate unit and stored in the partial sum buffer. Then, data from the buffer is loaded and processed by post-processing units (PPU) for Batch Normalization (BN), Rectified Linear Unit (ReLU), and pooling operations. Finally, the results for this layer are written back to the IFM through the write-back module. Upon the completion of the inference, the inference results stored in IFM are written to DDR.

2.2 Motivation

As previously mentioned, the memory wall issue remains unsolved in DCIM based accelerator [4] and quantization serves as an effective method for mitigating this issue [7]. When considering the CSR in CIM, Challenge 1 (Fig. 1) shows that the initial layers of the network are computation-bounded while the subsequent layers face memory-bounded limitations. As layers deepen,operations such as convolution and pooling make the size of IFM smaller, resulting in a decrease in computation time. On the other hand, the increase in the parameters of convolutional layers with deeper network layers leads to a rapid rise in storage time. For layers that are computation-bounded, low-bit-width quantization may not enhance their performance significantly. Therefore, we argue that adopting a hybrid quantization scheme for DCIM-PEA, considering both CSR and hessian trace to significantly improve energy efficiency while maintaining network accuracy. Furthermore, the inherent fault-tolerance capability of CNNs allows us to leverage approximate computation to enhance energy efficiency [14].

Secondly, constrained by parasitic effects on word and bit lines, DCIM typically adopts fine-grained MAC with large input channels to maximize energy efficiency, such as 1-bit multiplied by 4-bit and accumulated by 64 channels [5] or 2-bit multiplied by 8-bit and accumulated by 128 channels [6]. This design requires the utilization of shift and accumulate unit to execute high-bit-width MAC, which introduces extra power consumption. Moreover, the size of accumulation channel is limited. As depicted in Challenge 2 (Fig. 1), when the number of channels accumulated in CIM increases, power consumption exhibits a quadratic rise, imposing an upper bound on the number of channels. Thus, the shift-accumulate unit also undertakes the task of accumulating large channel convolutions. The above requirements render the power consumption of shift-accumulate unit non-negligible. Approximate computing has emerged as a potential solution since it can achieve a balance between accuracy and power consumption [10][11]. Besides, manual design methods are inefficient when facing wide design space, and

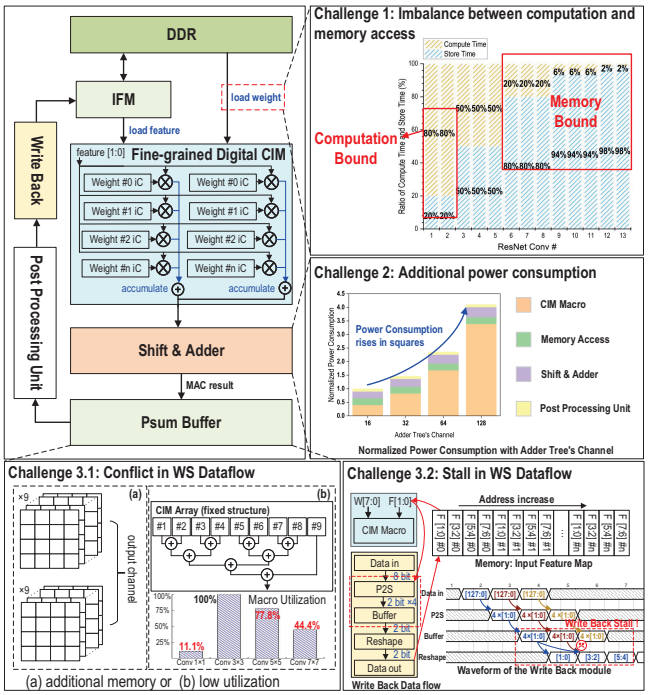


Figure 1: A typical DCIM based accelerator and three challenges remained to be solved

make it difficult to control error precisely. In this paper, CGP is utilized to generate approximate shift and accumulate unit to achieve a balance between energy-efficiency and accuracy.

Thirdly, as shown in Challenge 3.1 (Fig. 1), DCIM-based accelerators use WS dataflow mostly, which causes a conflict between the requirement for extra on-chip memory and low CIM utilization. Additionally, as illustrated in Challenge 3.2 (Fig. 1), the fine-grained write back inference results to the IFM causes the risk of dataflow stalls. Fine-grained DCIM macro presents optimization opportunities to address these challenges since it requires severl cycles to execute high-bit-width MAC.

3 FINE-GRAINED DIGITAL-CIM BASED CNN ACCELERATOR

The DCIM architecture employed in this paper is illustrated in Fig. 2 (a). The employed DCIM can simultaneously receive 128 2-bit features for MAC operation across 8 parallel row processing unit arrays. Based on the fine-grained characteristics of DCIM, we have comprehensively considered both software and hardware optimizations and proposed FDCA: Fine-grained Digital-CIM based CNN accelerator with hybrid quantization and weight-stationary dataflow, which is shown in Fig. 2. The hessian-trace-based quantization with approximate computing is first explained in Section 3.1. Afterwards, the CGP-based approximate shift and accumulate unit is proposed in Section 3.2. The remaining WS dataflow optimized for fine-grained DCIM is described in Section 3.3.

3.1 Hessian-Trace-Based Hybrid Quantization with Approximate Computing

This paper takes both hessian trace and CSR into consideration to implement layer-wise quantization to maximize performance

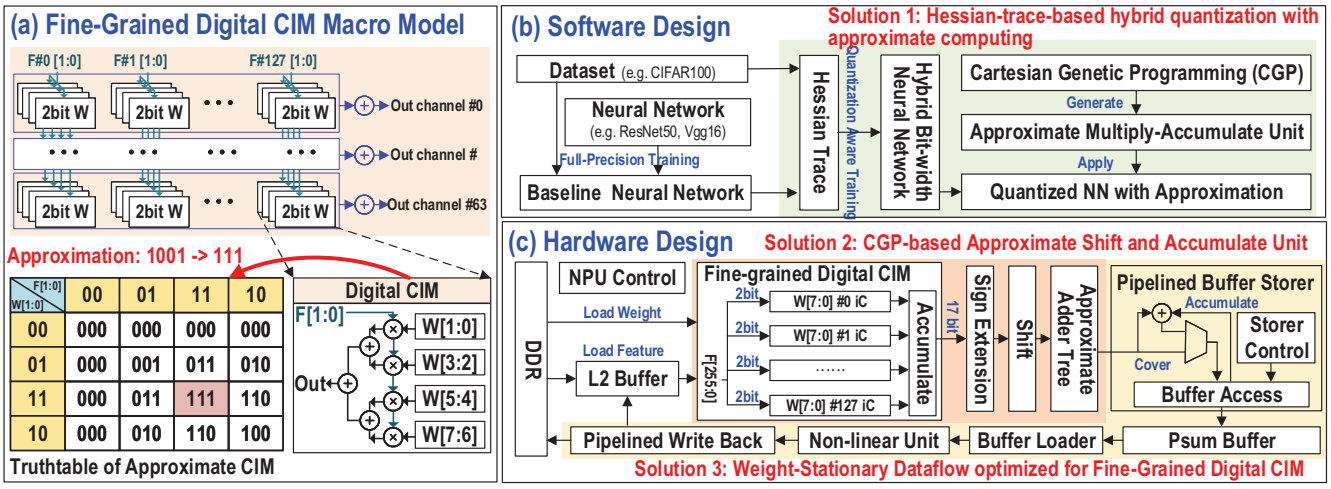


Figure 2: Employed fine-grained digital CIM macro model and proposed FDCA

while minimizing the accuracy loss. The CSR helps distinguish a compute-bound layer from a memory-bound one, and Hessian traces represent the layer's sensitivity to errors.

Given the employed DCIM architecture, the following formulas are employed to calculate storage time and computation time:

$$T_{storge} = N_c \cdot ceil(\frac{Ch_{in}}{128}) \cdot ceil(\frac{Ch_{out}}{64}) \cdot \frac{512}{32} \quad (1)$$

$$T_{comp} = N_c \cdot Oh \cdot Ow \cdot ceil(\frac{Ch_{in}}{128}) \cdot ceil(\frac{Ch_{out}}{8}) \cdot 8 \quad (2)$$

In Equation (1), the variable N_c denotes the number of convolution computations while Ch_{in} and Ch_{out} represent the input and output channels for each network layer. The *ceil* function is applied to round up to the nearest bigger integer number. In Equation (2), Oh and Ow denote the height and width of output, respectively.

The CSR for VGG16 and ResNet50 is indicated by the dashed line in Fig. 3(a) and Fig. 3(b), respectively. Typically, the number of output channels increases and the IFM becomes smaller as the network layers deepen in CNNs, leading to a significantly larger computation time compared to storage time in CIM. The presence of smaller IFMs results in the rapid refreshing of stored weights in CIM, while a higher number of weights contributes to an overall increase in memory access time. This difference causes the CSR to shrink, reaching only 1.54% in the final layers of ResNet50, rendering CIM highly inefficient.

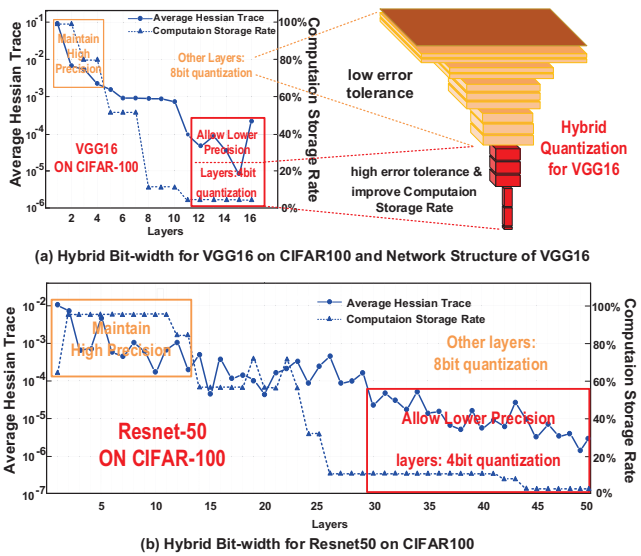
The Average Hessian Trace offers a more comprehensive representation of second-order information in CNNs and proficiently evaluates the sensitivity of each layer to errors [10]. A higher Average Hessian Trace indicates that the layer is more sensitive to errors rendering it unsuitable for lower-bit-quantization, while a lower value shows a higher error-tolerance.

The Average Hessian Traces of each layer in VGG16 and ResNet50 are respectively annotated with solid lines in Fig. 3(a) and 3(b). From the graphs, it can be observed that the hessian traces exhibit a similar trend to the CSR, wherein they are lager in the initial layers of the networks and smaller in the subsequent layers. This presents an opportunity for us to efficiently pursue hybrid quantization.

The inherent fault-tolerance capability of neural networks allows us to leverage approximate computation to further enhance computational efficiency. We quantify the errors introduced by approximate computation into offset values termed quantization errors, thereby implementing Quantization-Aware Training to ensure network accuracy.

By considering both the Average Hessian Traces and the CSR, our work performs layer-wise hybrid bit-width quantization on VGG16 and ResNet50. The first ten convolutional layers use 8-bit quantization for weights and features (8A8W), while the subsequent six convolutional layers use 4-bit quantization for weights and features (4A4W) in VGG16, which is shown in the right of Fig. 3 (a).

The quantization results are shown in Fig.4 For ResNet50, although substantial reductions in storage and computation time achieved through 4-bit quantization, the corresponding accuracy loss is too large as 10.29%. On the other hand, employing hybrid quantization yields a remarkable 45.71% reduction in storage time,



(a) Hybrid Bit-width for VGG16 on CIFAR100 and Network Structure of VGG16

(b) Hybrid Bit-width for Resnet50 on CIFAR100

Figure 3: Hybrid bit-width for VGG16 and Resnet50 considering average hessian trace and CSR

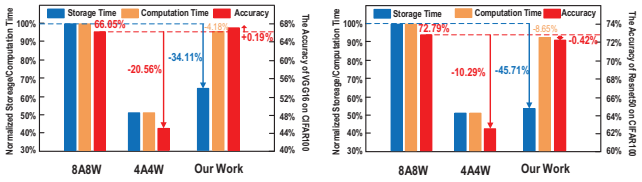


Figure 4: Energy efficiency improvement by using hybrid quantization

while maintaining accuracy. This effectively mitigates the memory wall issue. Similar observations hold true for VGG16. For VGG16, the precision achieved through proposed quantization surpasses even that of 8-bit quantization. This phenomenon can be attributed to the impact of mitigated impact of 4-bit quantization on the later layers during training, coupled with the more comprehensive training received by the front layers. As a result, the overall model benefits from a finer balance between reduced bit width and effective training across different layers.

3.2 CGP-Based Approximate Shift and Accumulate with Error Compensation

A variety of approximate logic synthesis methods have been proposed in recent years, among which CGP has been proved to be effective[15]. CGP automatically generates approximate circuits and evaluates their quality by calculating relative error distance (RED), thus upgrading circuit architecture by genetic and mutation algorithm. In this work, CGP-based approximate shift and accumulate unit is proposed to improve the energy-efficiency and compensate for errors caused by approximate multiplication.

Given the utilization of the previously described CIM, the input to the adder tree comprises 23 bits. Despite the error generated by approximate circuits, the network accuracy is not remarkably affected owing to the error tolerance of CNNs, and error is further reduced by inter-compensation of approximate multiplication utilized in DCIM. Accurate adders are adopted in the more significant bits and approximate adders are adopted in less significant bits of the adder tree, thereby striking a balance between accuracy and

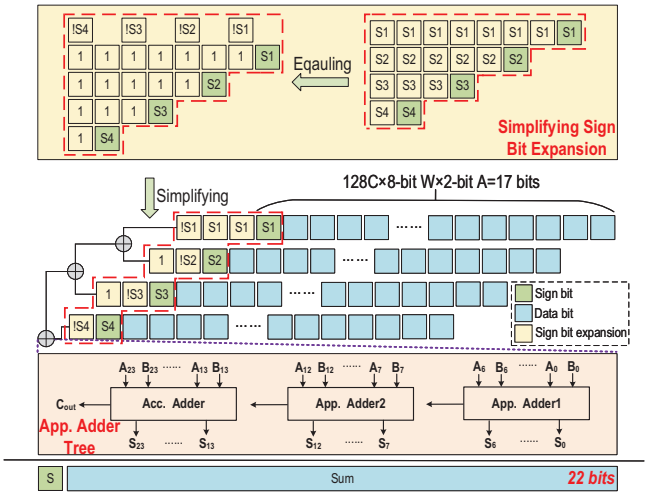


Figure 5: CGP-based approximate shift and accumulate unit

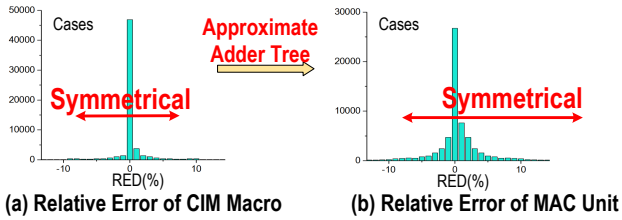


Figure 6: Relative error distribution of CIM and MAC unit

efficiency. Error bias in a specific direction can also be acquired by adding a penalty coefficient to the fitness function of CGP, thus generating circuits with target error. The CIM gets '111' instead of '1001' when calculating '11'×'11', which is illustrated in Fig. 2, making errors negative. An adder tree with positive error is needed to compensate for negative errors, reducing the overall error of approximate MAC unit.

The proposed approximate adder tree is illustrated in the Fig. 5. A simplified sign-bit expansion method is adopted in this work, which is illustrated in the upper part of Fig. 5. Assuming the sign bit of PP1 is S1, the 18th to 23rd are set to S1, which can be equivalently expressed as the expansion bits being set to 1 and added by the inverse of the sign bit. This implies that when S1 is 1, the expansion bits remain 1 after adding 0. If S1 is 0, the expansion bits become 0 after adding 1, noting that the carry bit of the 23rd overflows and is ignored.

As Fig. 6 shows, the CGP adders are generated based on the error generated by CIM to make the relative error distribution concentrated around 0 and close to symmetrical. The symmetrical error distribution can compensate the positive and negative erros during operation, thus improving inference accuracy [16]. Furthermore, the error of low bits has little impact on the results, so low-part OR adder (LOA) can be used for partial accumulation, which produces sum by an OR gate and has no carry [17]. LOA reduces both power and latency significantly due to its oversimplified structure.

Table 1: Comparison of Acc.Adder Tree and App.Adder Tree

	Acc. Adder Tree	App. Adder Tree	Improv.
Delay[1]	1.34 *ns*	0.61 *ns*	54.40%
Area[1]	65.46 μm^2	46.94 μm^2	28.30%
Power[1]	13.20 μW	9.10 μW	31.10%
PDP	17.69 *ns*-μW	5.55 *ns*-μW	68.63%
Accuracy loss	1.60%[2]	0.97%[3]	0.63%

[1]Synthesized using TSMC 28-nm under 0.8V, 400MHz
[2]tested with approximate CIM and accurate adder tree
[3]tested with approximate CIM and approximate adder tree

In proposed adder tree, accurate adders are adopted in the more significant bits, with CGP adders in the middle bits while LOA is used in the less significant bits of the adder tree to reduce hardware overhead with ignorable accuracy loss. The expansion sign bits are simplified in advance to reduce complexity as well. The experimental results illustrated in table I shows that the proposed approximate shift and accumulate unit achieves a 54.4% drop in delay while power consumption is reduced by 30%, making PDP reduced by 68.14%. The accurate adder tree and approximate adder tree are tested using ResNet50 on the CIFAR100. The results show that, owing to the error compensation mechanism, proposed approximate adder tree can mitigate the errors introduced by approximate

CIM. Specifically, the accuracy loss decreased from 1.60% to 0.97%, resulting in a recovery of 0.63%. The same experiment is also applied to VGG16 and the inference accuracy for VGG16 in CIFAR100 using approximate adder tree is 0.71%, improved by 0.12%.

3.3 Weight-Stationary Dataflow optimized for fine-grained Digital CIM

As mentioned above, the WS dataflow is an approach adopted mostly by CIM-based accelerators due to the inherent time-intensive nature of updating weights stored in the CIM macro[1][13]. However, there's a conflict to be solved between the low utilization of the CIM macro and requirement for extra memory when employing the WS dataflow, as previously highlighted. We propose the WS dataflow optimized for fine-grained digital CIM, as illustrated in Fig. 7. The proposed framework includes 3 parts, WS PEA for convolution, buffer storer for saving MAC results and pipelined writeback module for writing inference results back.

In the DCIM-PEA architecture, WS dataflow is employed as shown in Fig. 7(a). Convolution weights, indicated in green, are stored in CIM macro before the MAC operation and updated after the traversal of the IFM, which is denoted as yellow. The results of DCIM-PEA are further handled by the INT2FP, BN and FP2INT modules before being transmitted to the buffer storer. At each operation, CIM saves only one pixel of convolution weights with 128 input channels and 64 output channels. By this mapping method, convolution operations of any size, including fully connected network, can be transformed into 1×1 convolutions, making CIM macro be utilized at the maximum extent.

To minimum the data buffer and avoid extra memory, buffer storer for saving MAC results is used. As shown in Fig. 7(c), MAC data is initially buffered in a Data FIFO and the corresponding buffer address is generated by the control unit of data buffer storer, buffered in an address FIFO. The control unit determines the storage type based on whether the data represents the initial MAC result of convolution. If in cover mode, the result stored in the data FIFO will cover the data in data buffer at the corresponding address. Otherwise, the data in the data buffer at the corresponding address

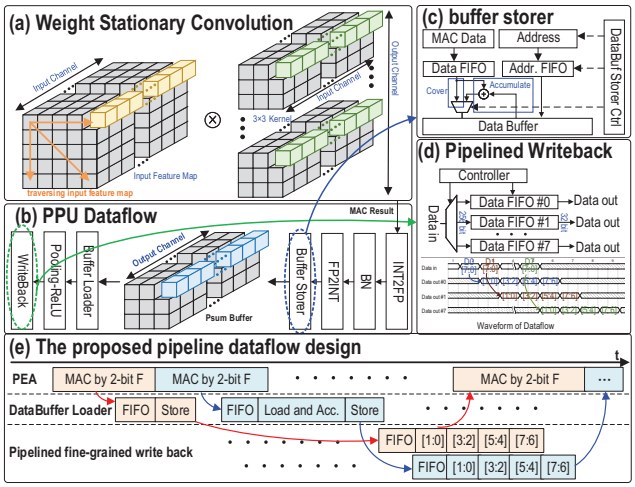


Figure 7: WS dataflow optimized for fine-grained DCIM

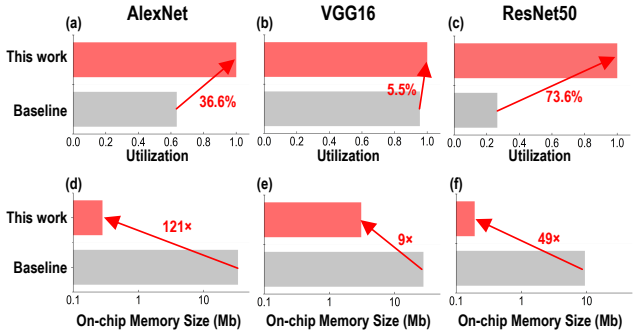


Figure 8: Utilization and memory size for baseline dataflow and optimized WS dataflow, (a)(d) AlexNet, (b)(e) VGG16, (c)(d) ResNet50

is loaded, accumulated with the result in data FIFO, and finally written back to the data buffer. This approach ensures utilization of the minimum data buffer capacity. Additionally, a pipelined writeback module is proposed to address potential stalls in the data flow during writeback of the IFM, which is shown in Fig. 7(d). In this module, the data to be saved in the IFM is buffered in a corresponding FIFO according to the address in the IFM, with 2-bit data being written each clock cycle.

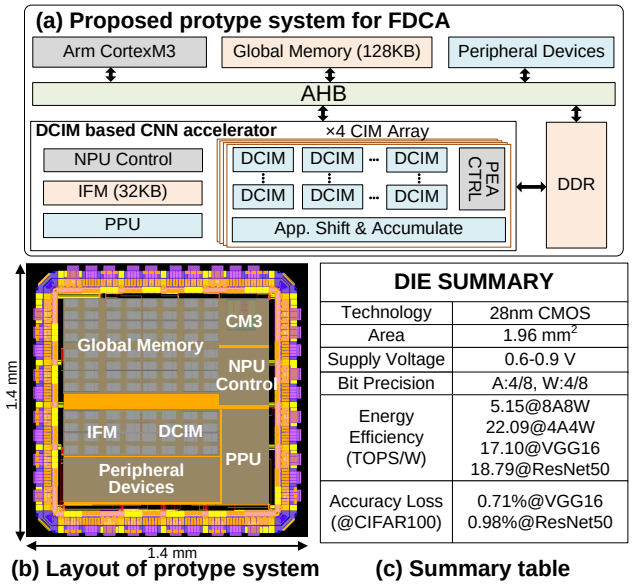
The entire pipeline dataflow is depicted in Fig. 7(e). In the DCIM-PEA system, a minimum of 4 cycles is necessary to execute an 8-bit activation multiplied by an 8-bit weight MAC operation. During this stall in PPU, the buffer storer can execute operation including detect, cover, and accumulation operations on the results from the preceding MAC round. Simultaneously, the pipelined writeback module utilizes FIFO buffers to temporarily store and subsequently write back the inference results to the IFM.

Experimental results are shown in Fig. 8. Fig. 8(a) (b) (c) shows that compared to memory-optimized WS dataflow, the proposed WS dataflow achieves 36.6%, 5.5%, and 73.6% utilization improvements. Compared to utilization-optimized WS dataflow, the proposed dataflow reduces memory size by 121×, 9×, and 49× for AlexNet, VGG16 and ResNet50 models, respectively, as shown in Fig. 8(d) (e) (f) .

4 EXPERIMENTAL RESULT

To evalute FDCA, we implemented a DCIM-based accelerator protype system using industry 28-nm technology. Fig. 9(a) shows the overall architecture, which includes a CortexM3 core, DCIM based CNN accelerator, global memory, DDR memory and peripheral devices. The DCIM based CNN accelerator includes 4 DCIM-PEA, NPU Control, IFM and PPU, supporting multi bit-width. The layout of protype system is shown in Fig. 9(b) with the die area of 1.96 mm^2. The total memory size in protype system is 192KB. The die characateristics are shown in summary table in Fig. 9(c). The proposed FDCA works under 0.6-0.9V and support 4/8 bit activation and 4/8 bit weight. The on-chip memory required in protype system is 64 KB, which is used as an input cache for accelerator. The protype system achieves 5.15TOPS/W@8A8W and 22.09TOPS/W@4A4W. Notably, the protype system achieves up to 17.1TOPS/W and 18.79TOPS/W with accuracy loss only 0.71% and 0.98% when running VGG16 and ResNet50 at CIFAR100, respectively.

Table 2 provides a comprehensive comparison between proposed FDCA and the state-of-the-art (SOTA). The implemented protype system achieves energy efficiency improvements of 1.76× and 1.29× compared to SOTA [19], respectively. Compared to SOTA, the proposed FDCA achieves higher energy efficiency with less accruacy loss. It is seen that our proposed design enhances the energy efficiency while maintainting high inference accruacy.



DIE SUMMARY

Technology	28nm CMOS
Area	1.96 mm^2
Supply Voltage	0.6-0.9 V
Bit Precision	A:4/8, W:4/8
Energy Efficiency (TOPS/W)	5.15@8A8W 22.09@4A4W 17.10@VGG16 18.79@ResNet50
Accuracy Loss (@CIFAR100)	0.71%@VGG16 0.98%@ResNet50

Figure 9: Overall architecture, layout and die characteristics of FDCA protype system

Table 2: Comparison of proposed FDCA with previous works

	DAC'20[18]	DAC'21[21]	ISSCC'21[19]	***This work***
Technique	28 nm	28 nm	65 nm	28 nm
Voltage	0.95-1 V	0.62-1.0 V	0.5-1.1 V	0.6-0.9 V
Frequency	-	12.05-151.2 MHz	25-100 MHz	250 MHz
Bit Precision	A:4/8 W:4/8	A:2 W:8	A:2/4/6/8 W:1-8	A:4/8 W:4/8
Energy Efficiency (TOPS/W)	3.2@8A8W	11.20@2A8W	6.20@VGG16 8.22@ResNet50	5.15@8A8W 22.09@4A4W ***17.10*@VGG16** ***18.79*@ResNet50**
Accuracy loss	0.5%@ResNet20[1]	1%@ResNet20[1] 2.5%@ResNet20[2]	2.24%@VGG16[1] 2.38%@ResNet50[1]	***0.71%@VGG16*[2]** ***0.98%@ResNet50*[2]**

[1] tested on CIFAR10 [2] tested on CIFAR100

5 CONCLUSION

In this paper, we present FDCA: a fine-grained digital-CIM based CNN accelerator with hybrid quantization and weight-stationary dataflow. A hessian-trace-based hybrid quantization for CNNs is proposed, which consider both hessian trace and CSR, reducing 34.11% and 45.71% storage time with 0.19% accuracy improvement and 0.42% loss for VGG16 and ResNet50 on CIFAR100, respectively. Besides, a CGP-based approximate shift and accumulate unit is used to compensate the error generated by CIM macro, achieving 68.63% reduction of PDP, and improving accuracy by 0.63%. Thirdly, a weight-stationary dataflow optimized for fine-grained digital CIM is proposed, achieving 36%, 5.5%, 73.6% improvement in utilization and 121×, 9×, 49× reduction in on-chip memory size for CIFAR100 using AlexNet, VGG16, and ResNet50, respectively. The experimental results demonstrate that under 28-nm process technology, when running VGG16 and ResNet50 on CIFAR100, the proposed FDCA achieves 17.1TOPS/W and 18.79TOPS/W with only a slight decrease in accuracy by 0.71% and 0.98%, respectively. Compared to the state-of-the-art, this represents the improvements of 1.76× and 1.29× with less accuracy loss.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China under Grant 2023YFB4403103, and the Fundamental Research Funds for the Central Universities under Grant 2242022k60009.

REFERENCES

[1] H. Kim et al., "Colonnade: A reconfigurable SRAM-based digital bit-serial compute-in-memory macro for processing neural networks," in IEEE JSSC, vol. 56, no. 7, pp. 2221-2233, 2021.
[2] F. Tu et al., "A 28nm 29.2TFLOPS/W BF16 and 36.5TOPS/W INT8 Reconfigurable Digital CIM Processor with Unified FP/INT Pipeline and Bitwise In-Memory Booth Multiplication for Cloud Deep Learning Acceleration," in ISSCC, vol. 56, pp. 1-3, 2022.
[3] G. Desoli et al., "16.7 A 40-310TOPS/W SRAM-based all-digital up to 4b in-memory computing multi-tiled NN accelerator in FD-SOI 18nm for deep-learning edge applications,"in ISSCC, pp. 260-262, 2023.
[4] H. Zhu et al., "COMB-MCM: Computing-on-memory-boundary NN processor with bipolar bitwise sparsity optimization for scalable multi-chiplet-module edge machine learning," in ISSCC, vol. 65, pp. 1-3, 2022.
[5] E.-J. Chang et al., "A 12-nm 0.62-1.61 mW Ultra-Low Power Digital CIM-based Deep-Learning System for End-to-End Always-on Vision," in VLSI, pp. 1-2, 2023.
[6] A. Guo et al., "A 28nm 64-kb 31.6-TFLOPS/W Digital-Domain Floating-Point-Computing-Unit and Double-Bit 6T-SRAM Computing-in-Memory Macro for Floating-Point CNNs," in ISSCC, pp. 128-130, 2023.
[7] G. Jedhe et al., "A 12nm 137 TOPS/W Digital Compute-In-Memory using Foundry 8T SRAM Bitcell supporting 16 Kernel Weight Sets for AI Edge Applications," in VLSI, pp. 1-2, 2023.
[8] C.-S. Lin et al., "A 48 TOPS and 20943 TOPS/W 512kb Computation-in-SRAM Macro for Highly Reconfigurable Ternary CNN Acceleration," in ASSCC, pp. 1-3, 2021.
[9] H. Wu et al., "Integer quantization for deep learning inference: Principles and empirical evaluation," arXiv: 2004.09602, 2020.
[10] Z. Dong et al., "Hawq: Hessian aware quantization of neural networks with mixed-precision," in ICCV, pp. 293-302, 2019.
[11] Y. Gong et al., "Quality driven systematic approximation for binary-weight neural network deployment," in TCAS-I, vol.69, no.7, pp. 2928-2940, 2022.
[12] B. Wang et al., "A 28nm Horizontal-Weight-Shift and Vertical-feature-Shift-Based Separate-WL 6T-SRAM Computation-in-Memory Unit-Macro for Edge Depthwise Neural-Networks," in ISSCC, pp. 134-136, 2023.
[13] W. Jiang et al.,"A 16nm 128kB high-density fully digital In Memory Compute macro with reverse SRAM pre-charge achieving 0.36 TOPs/mm2, 256kB/mm2 and 23.8TOPs/W," in ESSCIRC, pp. 409-412, 2023.
[14] H. Jiang et al., "Approximate arithmetic circuits: A survey, characterization, and recent applications," Proceedings of the IEEE, vol.108, no.12, pp. 2108-2135, 2020.
[15] L. Sekanina et al., "Evolutionary computing in approximate circuit design and optimization," in WAPCO 2015, pp.1-6, 2015.
[16] B. Liu et al., "An efficient BCNN deployment method using quality-aware approximate computing," in IEEE TCAD, vol. 41, no.11, pp. 4217-4228, 2022.
[17] H. R. Mahdiani et al., "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," TCAS-I, col. 57, no. 4, pp. 850-862, 2009.
[18] H. Jiang et al., "A Two-way SRAM Array based Accelerator for Deep Neural Network On-chip Training" in DAC, pp. 1-6, 2020.
[19] J. Yue et al., "15.2 A 2.75-to-75.9 TOPS/W computing-in-memory NN processor supporting set-associate block-wise zero skipping and ping-pong CIM with simultaneous computation and weight updating," in ISSCC, vol. 64, pp. 238-240, 2021.
[20] K. Ueyoshi et al., "DIANA: An end-to-end energy-efficient digital and analog hybrid neural network SoC," in ISSCC, vol. 65, pp. 1-3, 2022.
[21] K. Lee et al., "A charge-sharing based 8T SRAM in-memory computing for edge DNN acceleration," in DAC, pp. 739-744, 2021.