2022 2nd International Conference on Frontiers of Electronics, Information and Computation Technologies (ICFEICT) | 978-1-6654-5476-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICFEICT57213.2022.00100

A Near Memory Computing FPGA Architecture for Neural Network Acceleration

1st Guanchen Tao
Zhejiang University
Hangzhou, China
taoguanchen@gmail.com

2nd Yonggen Li
College of Information Science and Electronic Engineering
Zhejiang University
Hangzhou, China
12031011@zju.edu.cn

3rd Yanfeng Xu
China Electronics Technology Group Corporation
No.58 Research Institute
Wuxi, China
kakatm1997@sina.com

4th Jicong Fan
China Electronics Technology Group Corporation
No.58 Research Institute
Wuxi, China
fanjicong1988@126.com

5th Haibin Shen
College of Information Science and Electronic Engineering
Zhejiang University
Hangzhou, China
shen hb@zju.edu.cn

6th Kejie Huang*
College of Information Science and Electronic Engineering
Zhejiang University
Hangzhou, China
huangkejie@zju.edu.cn

***Abstract*—The Deep Learning Accelerators (DLAs) are gaining attention in recent years due to their advantages in efficiency, privacy, and bandwidth usage efficiency to operate deep neural networks. Field Programmable Gate Arrays (FPGAs) can offer low-power computation capacity, which is profound for the deployment of DLAs in AI edge computing devices. However, there are two major problems in deploying DLAs in FPGA: Firstly, processor and memory units in FPGA are separated, and the data transfer between them requires a large energy overhead. Secondly, the complexity of Deep Neural Network (DNN) models and the variety of FPGA platforms make it hard to design hardware accelerators when taking into account both performance and energy efficiency simultaneously. Near-Memory-Computing (NMC) has been a promising candidate to accelerate neural network computing. Therefore, this paper proposes an NMC FPGA architecture, which can be adapted to various neural network models by using Verilog-to-Routing (VTR) tool.**

Keywords- AI edge device; DNN accelerator; FPGA; Near- Memory-Computing; VTR; Toolchain;

I. INTRODUCTION

The rapid growth of Deep Neural Networks (DNNs) has gained great attention from various research areas. DNNs are data-centric, with millions of parameters to operate and transfer between memory and processor [1], [2], causing a large amount of energy consumption. To reduce high power overhead, the industry has turned to specialized hardware accelerators for DNN computations [3].

The main hardware platforms for DNN accelerators are Graphics Processing Unit (GPU), Application Specific Inte- grated Circuit (ASIC), and Field Programmable Gate Array(FPGA). The DNN acceleration with GPUs is limited by the underlying toolkit designed by the manufacturer. It also suffers from high power consumption and weak real-time performance. Although ASIC is relatively small in size and power consumption, it has the disadvantages of long design cycles, high development costs, and low agility. As for FPGA, although it has strong parallelism and high reconfigurability, its computing efficiency is still unsatisfying. To address these problems, we put forward the Near-Memory-Computing (NMC) FPGA platform for the DNN accelerator. Near-Memory-Computing (NMC), a promising computing paradigm to address the memory wall in conventional Von Neuman architecture, can significantly improve the energy efficiency of Multiply-Accumulate (MAC) operations in DNN accelerators [4], [5]. Multiple early research teams have proposed prototype for NMC [6]–[12]. With recent advance in die stacking technology, more specialized NMC systems including NMC DNN accelerator were developed [12]–[18]. Peripheral circuits for NMC accelerator are proposed in [19] to increase energy efficiency. However, existing work mainly focuses on the design of specialized NMC systems and overlooked the possibility of combining the energy efficiency of NMC with the agility of FPGA. Conventional FPGAs have rich BRAM IP resources on board, which can be utilized to develop SRAM-based NMC blocks to achieve high computing efficiency.

Nonetheless, using FPGAs to design accelerators still requires expertise. Several works [20], [21] have made efforts in utilizing FPGA accelerators, which have looked into ways to optimize memory footprint to match the memory bandwidth of particular FPGAs or optimize the operation execution order of DNN computation [3]. The optimization of these schemes is limited due to the ever-existing memory wall of Von Neumann's architecture. In this work, we develop various neural network components such as CONVOLUTION (CONV), FULLY-CONNECTED (FC), and POOLING (POOL). These components can be used to construct a complete neural

978-1-6654-5476-6/22/$31.00 ©2022 IEEE
DOI 10.1109/ICFEICT57213.2022.00100

network, which can be mapped to NMC FPGA using VTR [22] (an open-source Computer- Aided Design tool).

The contributions of the paper are summarized in the following:

- We propose an NMC-based FPGA architecture. An adder three is added beside BRAM to greatly improve the MAC computing speed.
- An Computer-Aided Design (CAD) tool modified from VTR [22] to automatically map the generated and synthesized neural network model to NMC-based FPGA and analyze its power usage, area, etc.

The rest of this article is structured as follows. Section II introduces the proposed NMC FPGA architecture. Section III presents the key components for neural network acceleration. Sections IV and V provide the method to map neural network layers to NMC FPGA and the evaluation results, respectively. Finally, Section V draws the conclusion.

II. NMC FPGA ARCHITECTURE

FPGAs provide users with the ability to reconfigure the hardware to meet specific needs. Common FPGA architecture consists of an array of logic blocks, called Configurable Logic Blocks (CLBs), I/O pads, and routing channels. Similar to the conventional FPGA architecture, our proposed NMC FPGA, as shown in Figure 1, adopts a typical island-style FPGA architecture. The IO Blocks (IOBs) of the overall FPGA are placed around the FPGA, and the basic computing resource is CLBs. Each CLB is composed of several LUTs and FFs in various interconnected ways. BRAM is the on-chip storage resource that is used to store intermediate calculation results. The core MAC calculation is completed by the NMC core distributed on the chip, and there are abundant routing resources between the logic blocks on the FPGA for placement and routing.

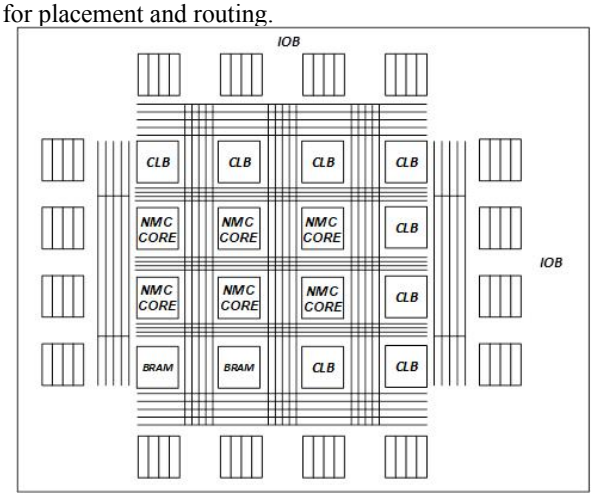


Figure 1. The block diagram of our proposed NMC FPGA architecture.

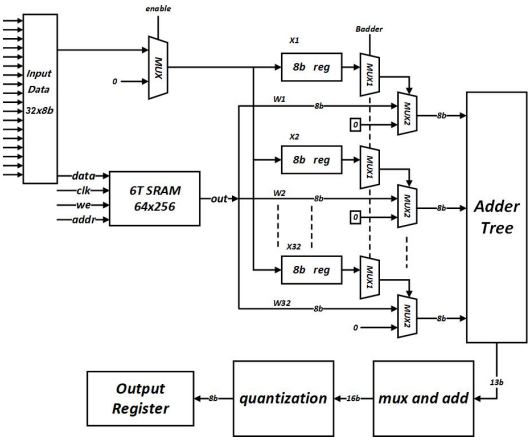


Figure 2. The block diagram of our proposed NMC core circuit.

The NMC core has high parallelism and scalability, which is capable of computing a mass of MAC operations simultaneously with much lower power consumption. As shown in Figure 2, the architecture of the NMC core consists of a SHIFT AND ADD block, an adder tree, a MUX AND ADD block, and a quantization block. Assuming both activations and weights are 8 bits, the product of activations and corresponding weights can be transformed into shift-adding as Equation 1:

$$d * w = \sum_{i=1}^{7} d[i] * w * 2^i \quad (1)$$

where d and w are 8-bit input data and weight, respectively. The weights are stored in the NMC BRAMs in advance, and the input feature map is stored in the non-NMC BRAMs. This circuit is capable of computing MAC calculation be- tween 64 8-bit weights and 64 8-bit inputs. The inputs are first transferred in the shift registers and multiplied with weight by the Badder one bit each cycle. The partial sum will be added in the adder tree to get the initial result, and the shift add will be accomplished in the subsequent MUX_ AND_ ADD module.

The Adder Tree consists of Ripple-Carry Adders (RCAs) as shown in Figure 3. It completes the addition of two multi- bit numbers by connecting the carry output and the carry input of two adjacent full adders. It performs the parallel computation of multiple numbers without additional registers to store partial sum. By dividing the number of addends into two groups and performing the addition operation, the obtained partial sum continues to be grouped in two groups, until the entire addition is completed. In addition to Adder Tree, a MUX AND ADD block is required to shift and add up the results of the Adder Tree according to their weights. Figure 4. illustrates the MAC operation of 2 vectors, and each vector has two 8-bit numbers.

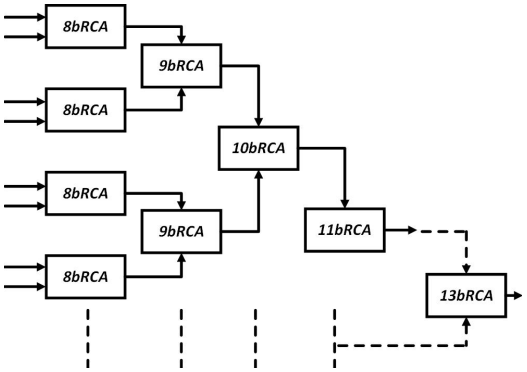


Figure 3. The block diagram of our proposed Adder Tree structure.

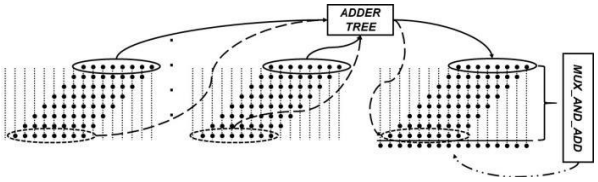


Figure 4. An example of the MAC operation with the proposed NMC cores.

III. KEY COMPONENTS FOR NEURAL NETWORK ACCELERATION

A. CONV AND FC-LAYER ACCELERATION

As mentioned above, the NMC core circuit is for parallel matrix-vector multiplication. The NMC core applies Weight-Stationary (WS) dataflow for both CONV and FC layers. The FC layer only requires storing the weights in the NMC core in advance, and then sending the input to the NMC core to conduct the vector-matrix computation. The dataflow of the CONV layers is different from the FC layers. In our design, IM2COL is adopted to transform the CONV computation into matrix-matrix multiplication. As depicted in Figure 5, the dimension of weight tensor for CONV layer is $H_f \times W_f \times C_i \times C_o$, where H_f is the filter height, W_f is the filter width, C_i is the number of input channels, and C_o is the number of output channels. After IM2COL transformation, all the weights can be stored in a $M \times C_O$ matrix, where $M = N_i \times H_o \times W_o$ and N_i is the number of the input feature map, H_o is the output height, and W_o is the output weight. Therefore, both FC layers and the CONV layers after IM2COL transformation perform Matrix- Vector-Multiplication (MVM), which can be formulated as $y = xW$, where x is the input matrix, and W is the weight matrix. We assume the size of NMC core is N_d N_w, where N_d is the core depth, and N_w is the core width. In most cases, a single NMC core is insufficient for computation of the whole layer, thus an NMC core array with $\frac{H_{num}}{N_d}$ rows and $\frac{W_{num}}{N_W}$ columns are required for FC and CONV layers.

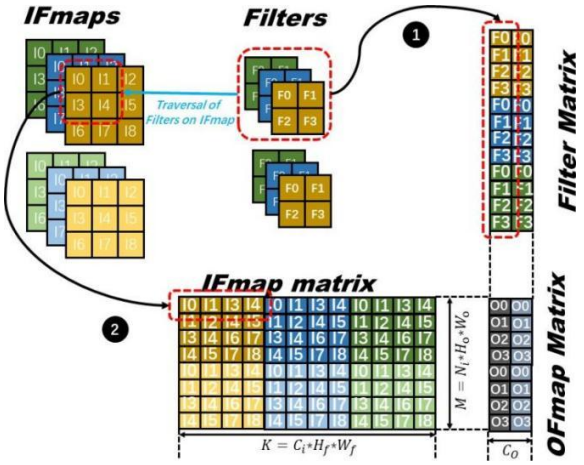


Figure 5. An illustration of IM2COL transformation.

B. POOLING-LAYER ACCELERATION

The architecture of the POOLING block is shown in Figure 6, which consists of two First-In-First-Out (FIFO) registers, a shift register, a dual-port comparator, a Serial-In-Parallel-Out (SIPO) block, and a controller. The functions of the major modules are introduced in the following:

The FIFO consists of a read pointer and a write pointer. Whenever there is a push signal, the write pointer increases by one, and when there is a pop signal, the read pointer increases. It is a RAM to store data. There are two FIFOs in this design: POOL FIFO saves the initial data of an entire channel input for the first round comparison in each POOLING window, as well as the intermediate comparison results; ROW FIFO saves the intermediate comparison result of the previous row in the POOLING window.

The shift register is used for the padding function of the POOLING layer. It is also served as a buffer between input data and the succeeding comparator module.

SIPO is the interface designed for connection between serial input and parallel output.

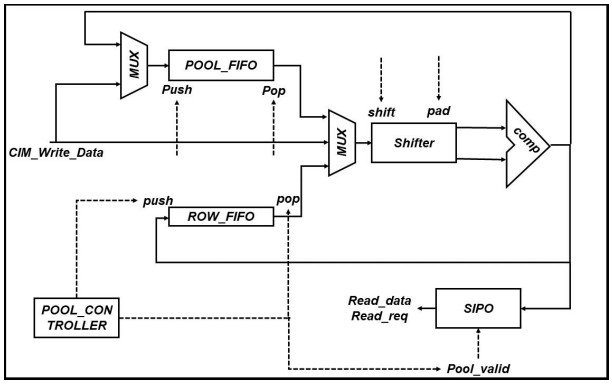
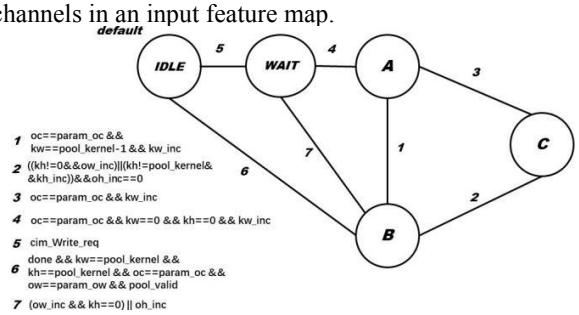


Figure 6. The block diagram of our proposed POOLING block

The comparator compares the two groups of serially input data sequentially to obtain the comparison result.

The controller is the kernel of the POOLING module, which consists of a Finite State Machine (FSM) and several counters that count in the dimensions of

Output Chanel (OC), Kernel Width(KW), Output Width(OW), Kernel Height(KH), and Output height(OH). The transition of states in FSM is shown in Figure 7(a). Figure 7(b) illustrates the state switching of different channels in an input feature map.



(a) Finite State Machine Transition

W	A	B	W	A	B
C	A	B	C	A	B
C	A	B	C	A	B
W	A	B	W	A	B
C	A	B	C	A	B
C	A	B	C	A	B

(b) Channel States transition on a 6*6input matrix and 3*3 POOLING kernel

Figure 7. FSM design

The dataflow of the POOLING layer is largely determinedby the dataflow of CONV layer, as POOLING layer serves as accessory after CONV layer. Algorithm 1 illustrates the output sequence of Conv layer.

Algorithm 1 Core output dataflow

```
Output sequence of Conv layer
for oc∈max output channel do
    oc++
    for kw∈max poolkernel width do
        kw++
        for ow∈ow max=max output width / poolkernel width do
            ow++
            for kh∈max poolkernel height do
                kh++
                for oh∈oh max=max output height / poolkernel height do
                    oh++
                    Push output from NMC CORE toPOOLING layer
                end for
            end for
        end for
    end for
end for
```

IV. MAPPING NEURAL NETWORK LAYERS TO FPGA

VTR (Verilog-to-Routing), as shown in Figure 8, is an open-source CAD project that is cooperating around the world. Odin II is responsible for front-end synthesis and partial mapping. Partial mapping is to search and match the model described in the HDL with the hard-core IP on the FPGA architecture. The logic not matched with any hard core IP is sent to ABC in the form of a netlist. ABC carries out logic optimization and technology mapping and then synthesizes and maps them into the Look-Up Table (LUT) and registers of the FPGA. Finally, VPR accepts the netlist output by ABC, performs packaging and layout according to the provided FPGA architecture, and then analyzes timing, area, and power consumption.

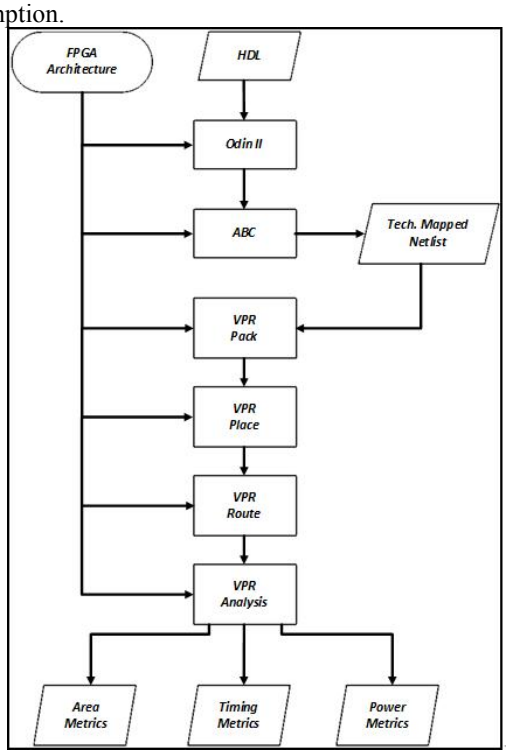


Figure 8. Components of VTR tool

To map the MAC circuit described in HDL to the NMC FPGA, it is necessary to declare the hard IP of the NMC core in the FPGA architecture description file, which defines the port type, the number of ports, port setup time, port hold time, dynamic and static power consumption of each port, and the layout information of the NMC cores. The CONV and FC layers will be mapped to an array of NMC cores, while for POOLING layers, the accelerator's logic will be transformed into netlist and mapped to LUTs and other hard cores eventually.

V. EVALUATION

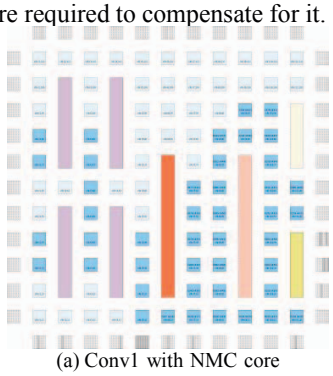
The performance of matrix multiplication regarding a row of activations and a column of weights (one MAC

calculation) is tested. Under the same condition of 40 nm process, 100 MHz clock frequency, and 1.1 V operating voltage, Table I shows the performance comparison of two circuits using the SYNOPSYS DESIGN Compiler. Our proposed NMC core reduces the power consumption of matrix-matrix multiplication by 2.875 × compared with the non-NMC version respectively. The area of the circuit is reduced by 64.11%, respectively.

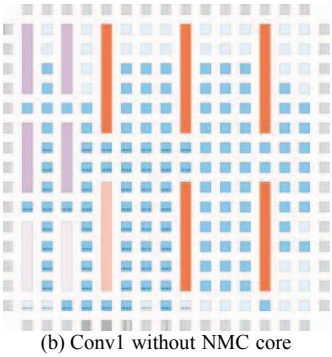
TABLE I. PERFORMANCE COMPARISON OF MAC COMPUTING ARRAYS

Circuit	Power	Area
NMC Core	$0.1281mW$	$1345.54um^2$
Systolic Array	$0.1922mW$	$4093.00um^2$

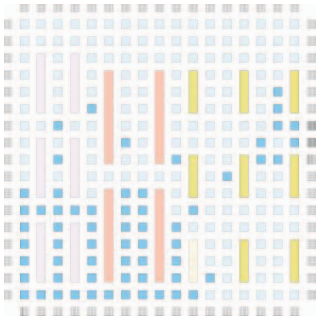
Furthermore, neural network acceleration for different layers has been mapped to both FPGA with and without NMC Figure 9. In Figure 9a, a single NMC core is invoked (yellow block), while in Figure 9b extra BRAMs (orange block) are invoked to make up for the NMC core. Aside from BRAMs and NMC core, the NMC FPGA proposed also contains abundant CLBs and Multipliers for mapping. Without an NMC core, more CLBs and Multipliers are required to compensate for it.



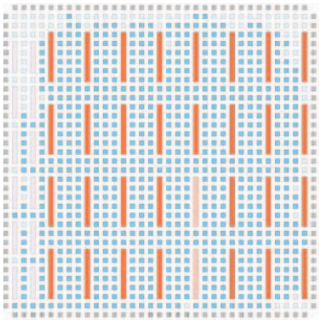
(a) Conv1 with NMC core



(b) Conv1 without NMC core



(c) FC with NMC core



(d) FC without NMC core

Figure 9. FPGA Layout After Mapping

As shown in Table II, DNN layers including Conv1, Conv3, Conv5, and FC are tested. Conv1, Conv3, and Conv5 differ in their input and kernel sizes, where Conv1 is the smallest and Conv5 is the largest. Our proposed NMC FPGA improves the energy efficiency by 1.52× ~ 1.7× when compared with the non-NMC one. The area is also greatly reduced to map the same size network layers.

TABLE II. COMPARISON OF DNN ACCELERATION WITH NMC FPGA AND NORMAL FPGA

Layer type	# of NMC cores	Total Power (W)	Area	Energy Efficiency (GTOPS/W)
Conv1	1	0.0098173	13*13	81.49
Conv3	5	0.0305345	19*19	131.00
Conv5	26	0.1392998	28*28	149.32
FC	8	0.0460724	19*19	138.91
Conv1 no NMC	0	0.01166	17*17	53.6
Conv3 no NMC	0	0.04862	31*31	77.1
Conv5 no NMC	0	0.1865	62*62	107.2
FC no NMC	0	0.07131	35*35	84.1

VI. CONCLUSION

A high-efficiency NMC FPGA architecture has been pro- posed in this paper, which aims to greatly reduce the power consumption of the matrix multiplication. With NMC FPGA, the matrix computing energy efficiency has been boosted by 1.52×–1.7×. Our proposed NMC core has reduced the power consumption of CONV by 1.58× when compared with the non-NMC version. The area of the circuit has also been reduced by 64.11%.

REFERENCES

[1] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation net- works," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 7132–7141.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[3] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An efficient hardware accelerator for structured sparse con- volutional neural networks on fpgas," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 9, pp. 1953–1965, 2020.

[4] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra, "A review of near-memory computing architectures: Opportunities and challenges," in 2018 21st Euromicro Conference on Digital System Design (DSD). IEEE, 2018, pp. 608–617.

[5] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans, H. Corporaal, and A. J. Boonstra, "Near-memory computing: Past, present, and future," Microprocessors and Microsystems, vol. 71, p. 102868, 2019.

[6] P. M. Kogge, "Execube-a new architecture for scaleable mpps," in 1994 International Conference on Parallel Pro- cessing Vol. 1, vol. 1. IEEE, 1994, pp. 77–84.

[7] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Kee- ton, C. Kozyrakis, R. Thomas, and K. Yelick, "A case for intelligent ram," IEEE micro, vol. 17, no. 2, pp. 34–44, 1997.

[8] D. Patterson, T. Anderson, and Cardwell, "Intelligent ram (iram): chips that remember and compute," in 1997 IEEE In- ternational Solids-State Circuits Conference. Digest of Tech- nical Papers, 1997, pp. 224–225.

[9] M. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, J. Brockman, A. Srivastava et al., "Mapping irregular applications to diva, a pim-based data- intensive architecture," in SC'99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing. IEEE, 1999, pp. 57–57.

[10] Y. Kang, W. Huang, S.-M. Yoo, D. Keen, Z. Ge, V. Lam, P. Pattnaik, and J. Torrellas, "Flexram: Toward an advanced intelligent memory system," in Proceedings 1999 IEEE Inter- national conference on computer design: VLSI in computers and processors (Cat. No. 99CB37040). IEEE, 1999, pp. 192–201.

[11] D. G. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, and R. McKenzie, "Computational ram: Implementing processors in memory," IEEE Design & Test of Computers, vol. 16, no. 1, pp. 32–41, 1999.

[12] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart memories: A modular reconfigurable architecture," in Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No. RS00201). IEEE, 2000, pp. 161–171.

[13] I. Fernandez, R. Quislant, C. Giannoula, M. Alser, J. Go´mez-Luna, E. Gutie´rrez, O. Plata, and O. Mutlu, "Exploiting near- data processing to accelerate time series analysis," arXiv preprint arXiv:2206.00938, 2022.

[14] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a micro-46 workshop," IEEE Micro, vol. 34, no. 4, pp. 36–42, 2014.

[15] Q. Zhu, T. Graf, H. E. Sumbul, L. Pileggi, and F. Franchetti, "Accelerating sparse matrix-matrix multiplication with 3d- stacked logic-in-memory hardware," in 2013 IEEE High Per- formance Extreme Computing Conference (HPEC). IEEE, 2013, pp. 1–6.

[16] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, "Drama: An architecture for accelerated processing near memory," IEEE Computer Architecture Letters, vol. 14, no. 1, pp. 26–29, 2014.

[17] Farmahini, A. Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, "Nda: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules," in 2015 IEEE 21st International Symposium on High Perfor- mance Computer Architecture (HPCA). IEEE, 2015, pp. 283–295.

[18] P. Das and H. K. Kapoor, "Clu: A near-memory accelerator exploiting the parallelism in convolutional neural networks," ACM Journal on Emerging Technologies in Computing Sys- tems (JETC), vol. 17, no. 2, pp. 1–25, 2021.

[19] R. Xiao, K. Huang, and H. Shen, "An overview of computing- in-memory interfaces," in International Conference on Fron- tiers of Electronics, Information and Computation Technolo- gies, 2021, pp. 1–6.

[20] J. Liu and K. Huang, "A novel scheme to map convo- lutional networks to network-on-chip with computing-in- memory nodes," in 2020 International SoC Design Confer- ence (ISOCC), 2020, pp. 126–127.

[21] H. Sharma, J. Park, E. Amaro, B. Thwaites, P. Kotha, A. Gupta, J. K. Kim, A. Mishra, and H. Esmaeilzadeh, "Dnnweaver: From high-level deep network models to fpga acceleration," in the Workshop on Cognitive Architectures, 2016.

[22] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. El- dafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. Walker et al., "Vtr 8: High-performance cad and customizable fpga architecture modelling," ACM Transactions on Reconfig- urable Technology and Systems (TRETS), vol. 13, no. 2, pp. 1–55, 2020.