安徽建筑大学
ANHUI JIANZHU UNIVERSITY

英 文 翻 译

姓　　名：年志豪

学　　号：20210040216

学　　院：电子与信息工程学院

专　　业：电子信息工程

完成时间：2024 年 4 月 19 日

近内存计算 FPGA 神经网络加速器架构

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

摘要

由于深度学习加速器（DLA）在运行深度神经网络时具有效率、隐私性和带宽利用率方面的优势，近年来它们受到了关注。现场可编程门阵列（FPGA）可以提供低功耗计算能力，这对部署边缘人工智能设备中的DLA至关重要。然而，在FPGA中部署DLA存在两个主要问题：首先，FPGA中的处理器和存储单元分离，它们之间的数据传输需要大量的能耗开销；其次，深度神经网络模型的复杂性以及FPGA平台的多样性使得同时考虑性能和能效进行硬件加速器设计变得困难。近内存计算（NMC）一直是加速神经网络计算的有希望的选择。因此，本文提出了一种基于Verilog-to-Routing（VTR）工具的NMC FPGA架构，该架构可以适应各种神经网络模型。

关键词：边缘人工智能设备；深度神经网络加速器；现场可编程门阵列；近内存计算；VTR；工具链

I. 引言

深度神经网络（DNN）的快速增长引起了各个研究领域的广泛关注。 DNN以数据为中心，具有数百万个参数需要在内存和处理器之间操作和传输，导致大量能源消耗。为了降低高功耗开销，行业已转向专用硬件加速器来执行 DNN 计算[3]。 DNN 加速器的主要硬件平台，它们分别是图形处理器（GPU）、专用集成电路（ASIC）和现场可编程门阵列（FPGA）。 GPU加速深度神经网络受限于制造商设计的底层工具包。它还受到高功耗的影响，由于其巨大的内存消耗和低实时性能，专用集成电路ASIC 虽然在尺寸和功耗上有优势，但设计周期长、开发成本高、灵活性差。现场可编程门阵列FPGA虽然具有强大的并行处理能力和高度的重构性，但其计算效率仍然不令人满意。为了解决这些问题，我们提出了基于近内存计算（NMC）的FPGA平台用于深度神经网络加速器。近内存计算是一种有前景的计算范例，可以显著提高传统冯·诺依曼体系结构中的内存墙问题的乘积累加（MAC）操作能效[4][5]。多个早期研究团队已经为NMC提出了原型[6]-[12]。随着堆叠芯片技术的最新进展，已经开发出了包括NMC DNN加速器在内的更多专用NMC系统[12]-[18]。为了提高能源利用效率，提出了NMC加速器的周边电路[19]。然而，目前的工作主要集中在专用NMC系统的开发上，而忽略了将NMC的能源效率与FPGA的敏捷性相结合的可能性。传统的FPGA具有丰富的嵌入式块存取存储器（BRAM）资源，可用于开发基于SRAM的NMC块以实现高效的计算。

然而，使用现场可编程门阵列(FPGA)设计加速器仍然需要专业知识。[20][21]已经对利用FPGA加速器进行了研究，研究了如何优化内存占用以匹配特定FPGA的内存带宽或优化DNN计算的运算执行顺序。[3] 由于冯·诺伊曼体系结构中永远存在的内存墙，这些方案的优化受到限制。 在这项工作中，我们开发了各种神经网络组件，如 卷积（CONV）， 全连接（FC）和 池化（POOL）。 这些组件可用于构建完整的神经网络，可以使用VTR[22]（一种开源的计算机辅助设计工具）映射到NMC FPGA。本文的贡献总结如下：

我们提出了一种基于NMC的FPGA架构。在BRAM旁边添加了一个三加法器，极大地提高了MAC计算速度。

从 VTR [22] 修改而来的计算机辅助设计 (CAD) 工具，用于自动映射生成和综合的神经网络模型到基于 NMC 的 FPGA 并分析其功耗、面积等。

本文其余部分的结构如下。第 II 部分介绍所提议的 NMC FPGA 架构。第三部分介绍了神经网络加速的关键组件。第四和第五部分提供了将神经网络层映射到 NMC FPGA 的方法以及相应的评估结果。最后，第五部分得出结论。

II. NMC FPGA 架构

现场可编程门阵列(FPGA)为用户提供了重新配置硬件以满足特定需求的能力。常见的FPGA架构由逻辑块阵列、输入/输出垫片和互连通道组成。与传统的FPGA架构类似，我们的提出的NMC FPGA采用了一种典型的岛屿式FPGA架构，如图1所示。整个FPGA的I/O块放置在FPGA周围，基本计算资源是CLB。每个CLB由多个LUTs和FFs以不同的方式互联而成。BRAM是一种用于存储中间计算结果的片上存储资源。核心MAC计算由芯片上的NMC核心完成，并且FPGA上的逻辑块之间有丰富的路由资源可用于放置和路由。

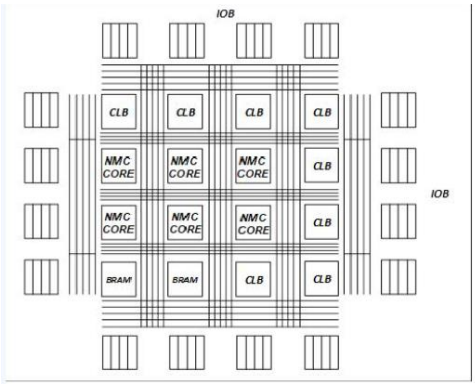


图 1.我们提出的 NMC FPGA 架构的框图

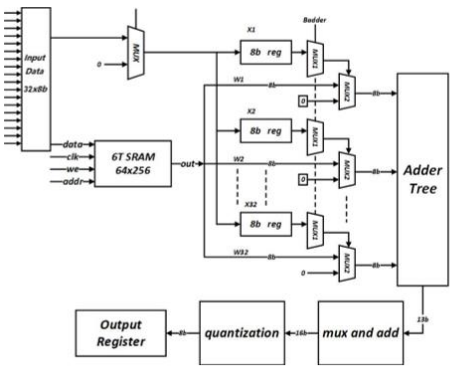


图 2.我们提出的 NMC 核心电路的框图

NMC 核心具有很高的并行性和可扩展性，能够在更低的功耗下同时计算大量乘加操作。如图 2 所示，NMC 核心由移位求和块、加法器树、Mux 和加法块以及量化块组成。假设激活和权重都是 8 位，则可以将激活值与相应权重的乘积转换为移位加法，如（1）所示：

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

其中d和w分别是8位输入数据和权重。权重预先存储在NMC BRAM中，而输入特征图存储在非NMC BRAM中。该电路能够计算 64 个 8 位权重与 64 个 8

位输入之间的 MAC 计算。输入首先通过移位寄存器，然后通过 Badger 每个周期乘以一个权重。部分求和将在加法树中相加以获得初始结果，并且随后将在 MUX_ AND_ ADD 模块中执行移位加法。

Adder Tree 由多个 ripple-carry adders (RCAs) 组成，如图 3 所示。它通过连接两个相邻全加器的进位输出和进位输入来完成两个多位数的相加。在没有额外寄存器来存储中间结果的情况下，它可以并行计算多个数字的总和。通过将加数分成两组并执行加法运算，得到的部分和继续分组，直到整个加法完成。除了 Adder Tree 之外，还需要一个 MUX AND ADD 块来根据权重对 Adder Tree 的结果进行移位和求和。图 4 显示了两个向量的乘积累加操作，每个向量包含两个 8 位数字。

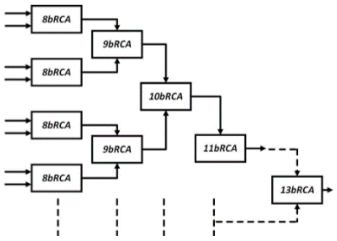


图 3. 我们提出的加法器树状结构框图

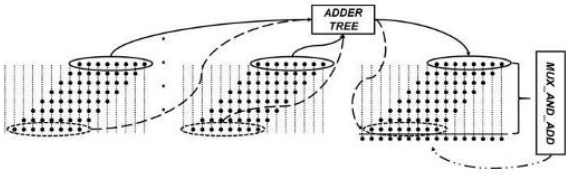


图 4. 预想的 NMC 核心的 MAC 操作示例

III. 神经网络加速的关键要素

A.卷积层和全连接层加速

如上文所述，NMC 核心电路用于并行矩阵-向量乘法。NMC 核心使用 Weight-Stationary (WS) 数据流为卷积层和全连接层。全连接层仅需要在核心中预先存储权重，然后将输入发送到核心以进行向量-矩阵计算。卷积层的数据流程与全连接层不同。在我们的设计中，采用 Im2Col 将卷积运算转换为矩阵-矩阵相乘。如图 5 所示，卷积层的权重张量维度为 Hf×Wf×Ci×Co，其中 Hf 是滤波器的高度，Wf 是滤波器的宽度，Ci 是输入通道数，而 Co 是输出通道数。经过 Im2Col 转换后，所有权重都可以存储在一个 M×CO 矩阵中，其中 M = Ni × Ho × Wo，Ni 是输入特征图的数量，Ho 是输出高度，而 Wo 是输出宽度。因此，FC 层和 Im2Col 转换后的卷积层都执行矩阵-向量乘法（MVM），可以表示为 y = xW，其中 x 是输入矩阵，W 是权重矩阵。我们假设 NMC 核心的大小为 Nd Nw，其中 Nd 是核心深度，Nw 是核心宽度。在大多数情况下，单个 NMC

核心不足以对整个层进行计算，因此需要具有 Hn- 行和 Wn- 列的 NMC 核心数组来处理 FC 和卷积层。

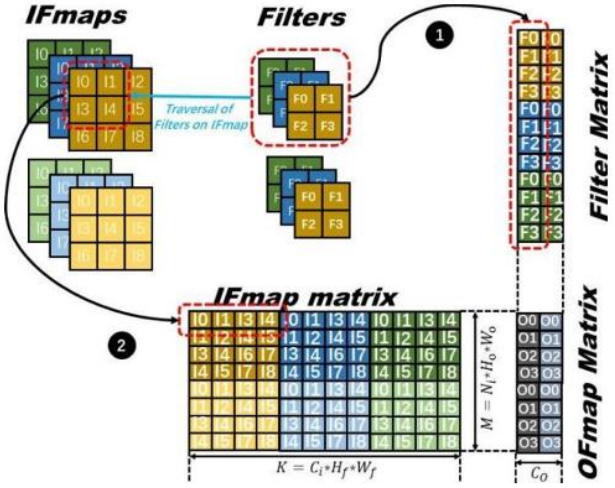


Figure 5. An illustration of IM2COL transformation.

图 5. 卷积层到矩阵转换的示意图

B. 聚合层加速

图6显示了POOLING块的架构，它由两个先进先出（FIFO）寄存器、移位寄存器、双端口比较器、串行输入并行输出（SIPO）模块和控制器组成。以下是主要模块的功能介绍：

FIFO 由读指针和写指针组成。每当有一个推信号，写指针增加一；当有一个弹出信号时，读指针增加。它是一种用于存储数据的随机存取存储器。在这个设计中，有两个FIFO：POOL FIFO 保存了每个池窗口中的整个通道输入的初始数据以及中间比较结果；ROW FIFO 保存了池窗口中上一行的中间比较结果。

移位寄存器用于分组层的填充功能。它还充当输入数据与后续比较模块之间的缓冲区。

SIPO 是专为串行输入与并行输出之间连接而设计的接口。

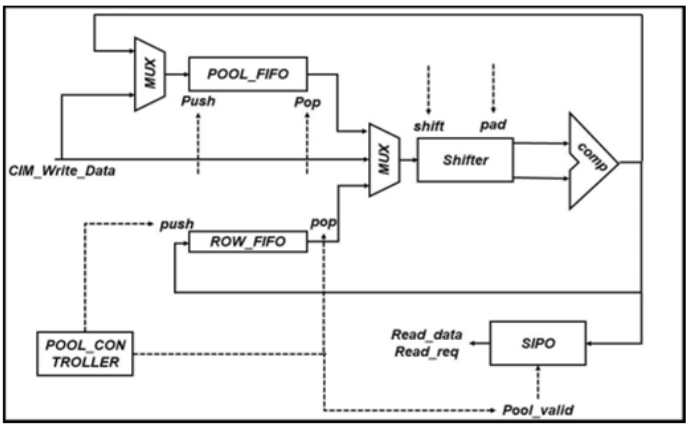
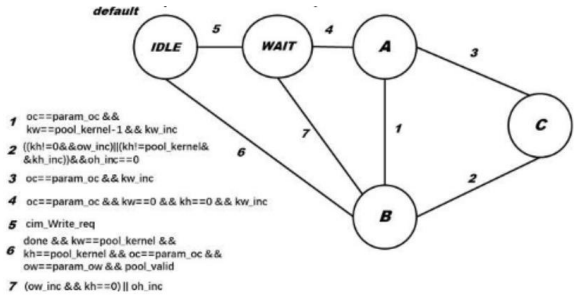


Figure 6. The block diagram of our proposed POOLING block

图 6. 我们提出的 POOLING 块的框图

比较器按顺序逐位输入数据组进行比较，以获得比较结果。控制器是POOLING模块的核心部分，它包括一个有限状态机（FSM）以及多个分别针对输出通道（Output Chanel，OC）、卷积核宽度（Kernel Width，KW）、输出宽度（Output Width，OW）、卷积核高度（Kernel Height，KH）和输出高度（Output Height，OH）进行计数的计数器。图7(a)展示了FSM中各状态之间的转换关系。而图7(b)则形象地说明了输入特征图中不同通道之间的状态切换过程。



(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

(a) 有限状态机转换　　(b) 通道状态

图 7.有限状态机设计

由于池化层在卷积层之后，所以池化层的数据流主要由卷积层的数据流决定。算法1显示了卷积层的输出序列。

Algorithm 1 Core output dataflow

```
1: Output sequence of Conv
layer2: for oc∈max output channel do 3: oc++
4:     for kw∈max poolkernel width do
5:         kw++
6:         for ow∈ow max=max output width / poolkernel width do
7:             ow++
8:             for kh∈max poolkernel height do
9:                 kh++
10:                for oh∈oh max=max output height / poolkernel height do
11:                    oh++
12:                    Push output from NMC CORE toPOOLING layer
13:                end for
14:            end for
15:        end for
16:    end for
17: end for
```

IV. 将神经网络层映射到 FPGA

VTR (Verilog-to-Routing)，如图8所示，是一个全球合作的开源EDA(电子设计自动化)项目。Odin II 负责前端综合和部分映射。部分映射是在FPGA架构中搜索并匹配 HDL(硬件描述语言) 中描述的模型与硬核IP(知识产权核)。未与任何硬核IP匹配的逻辑以网表的形式发送到ABC。ABC执行逻辑优化、技术和布局映射，然后将其综合并映射到FPGA的查找表（LUT）和寄存器。最后，VPR接受来自ABC的输出网表，根据提供的FPGA架构进行封装和布局，然后分析时序、面积和功耗。

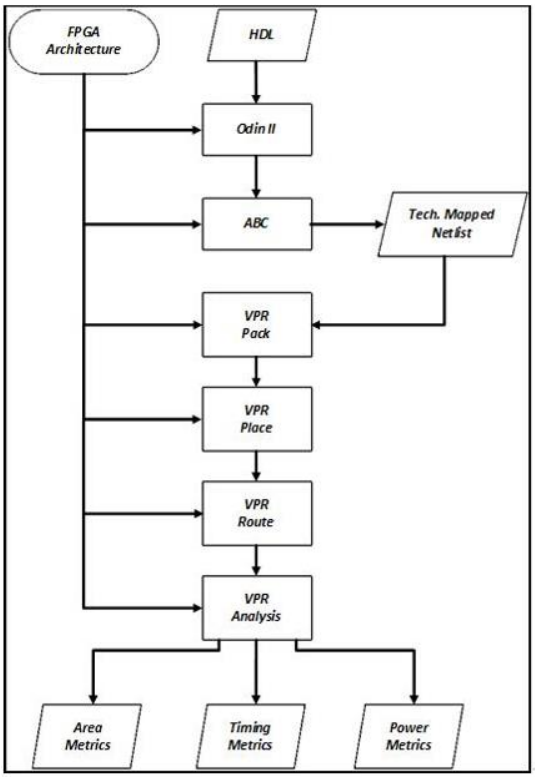


图 8. VTR 工具组件

为了在 HDL 中描述的 MAC 电路映射到 NMC FPGA，需要在 FPGA 架构描述文件中声明 NMC 核心的硬 IP。它定义了端口类型、端口数量、端口设置时间、端口保持时间、每个端口的动态和静态功耗以及 NMC 核心的布局信息。CONV 和 FC 层将映射到一个 NMC 核心阵列，而 POOLING 层则将加速器逻辑转换为网表并最终映射到 LUT 和其他硬核上。

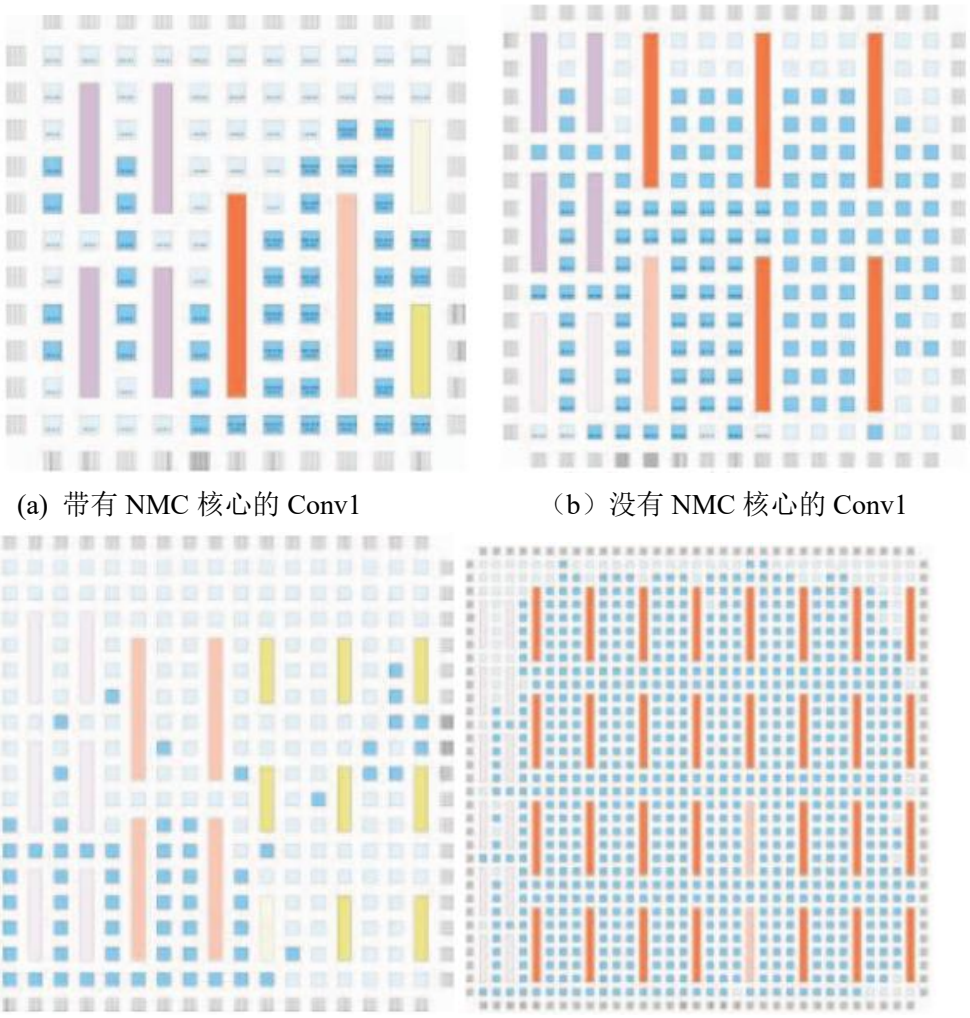
Ⅴ. 评价

在相同的40纳米工艺、100兆赫兹时钟频率和1.1伏工作电压条件下，我们测试了关于一行激活值与一列权重（一次乘加运算，MAC计算）的矩阵乘法性能。表I展示了使用SYNOPSYS DESIGN Compiler进行的两种电路性能对比。我们提出的NMC核心相较于非NMC版本，在矩阵-矩阵乘法运算上能够将功耗降低2.875倍。与此同时，该电路的面积也减少了64.11%。

表 1. Mac 计算阵列性能比较

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

此外，不同层的神经网络加速已经被映射到带或不带 NMC 的 FPGA 图 9。在图 9(a) 中，单个 NMC 核被调用（黄色方块），而在图 9(b) 中，额外的 BRAM 被调用（橙色方块）以弥补 NMC 核。除了 BRAM 和 NMC 核外，所提议的 NMC FPGA 还包含丰富的 CLB 和乘法器用于映射。如果没有 NMC 核，则需要更多的 CLB 和乘法器来补偿它。



(a) 带有 NMC 核心的 Conv1　　（b）没有 NMC 核心的 Conv1

(c) 带有 NMC 核心的 FC　　(d) 没有 NMC 核心的 FC

图 9.布线后 FPGA 布局

如表二所示，测试了包括Conv1、Conv3、Conv5和FC在内的DNN层。Conv1、Conv3和Conv5在输入大小和内核大小上有所不同，其中Conv1最小，而Conv5最大。我们提出的NMC FPGA与非NMC版本相比，在相同大小的网络层中可以提高能效1.52倍到1.7倍。面积也大大减小。

表二. DNN 加速与 NMC FPGA 和普通 FPGA 的比较

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. 结论

本文提出了一种高效的NMC FPGA架构，旨在大大降低矩阵乘法的功耗。通过使用NMC FPGA，矩阵计算的能量效率提高了1.52至1.7倍。与非NMC版本相比，我们提出的NMC核心使CONV的功耗降低了1.58倍。电路面积也减少了64.11%。

参考文献

[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 201821st 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,” in1994 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.