Machine Learning Training on a Memory-Centric Computing System

Juan Gómez-Luna[1] Yuxin Guo[1] Sylvan Brocard[2] Julien Legriel[2] Remy Cimadomo[2]
Geraldo F. Oliveira[1] Gagandeep Singh[1] Onur Mutlu[1]
[1]ETH Zürich [2]UPMEM

***Abstract*—Training machine learning (ML) algorithms is a computationally intensive process, which is frequently memory-bound due to repeatedly accessing large training datasets. As a result, processor-centric systems (CPU, GPU) waste large amounts of energy and execution cycles due to the data movement between memory units and processing units. Memory-centric computing systems, i.e., systems with processing-in-memory (PIM) capabilities, can alleviate this data movement bottleneck.**

Our goal is to understand the potential of general-purpose PIM architectures to accelerate ML training. To do so, we (1) implement several classic ML algorithms (namely, linear regression, logistic regression, decision tree, K-Means clustering) on a real-world general-purpose PIM architecture, (2) evaluate and characterize them in terms of accuracy, performance and scaling, and (3) compare to their counterpart state-of-the-art implementations on CPU and GPU. Our evaluation on a real memory-centric computing system with more than 2500 PIM cores shows that PIM greatly accelerates memory-bound ML workloads, when the necessary operations and datatypes are natively supported by PIM hardware. For example, our PIM implementation of decision tree is $27\times$ faster than the CPU implementation on an 8-core Intel Xeon, and $1.34\times$ faster than the GPU implementation on an NVIDIA A100. Our K-Means clustering is $2.8\times$ and $3.2\times$ than CPU and GPU implementations, respectively. This short paper presents several key observations, takeaways, and recommendations for users of ML workloads, programmers of PIM architectures, and hardware designers and architects of future memory-centric computing systems. More details about our work are available in [1].

***Index Terms*—processing-in-memory, processing-near-memory, machine learning, linear regression, logistic regression, decision tree, K-Means**

I. Introduction

Machine learning (ML) algorithms [2–7] have become ubiquitous in many fields of science and technology due to their ability to learn from and improve with experience with minimal human intervention. These algorithms train by updating their model parameters in an iterative manner to improve the overall prediction accuracy. However, training machine learning algorithms is a computationally intensive process, which requires large amounts of training data. Accessing training data in current processor-centric systems (e.g., CPU, GPU) implies costly data movement between memory and processors, which results in high energy consumption and a large percentage of the total execution cycles. This data movement can become the bottleneck of the training process, if there is not enough computation and locality to amortize its cost.

One way to alleviate the cost of data movement is *processing-in-memory* (*PIM*) [8–12], a data-centric computing paradigm that places processing elements near or inside the memory arrays. PIM has been explored for decades [10, 13–147]. However, memory technology challenges prevented from its successful materialization in commercial products. For example, the limited number of metal layers in DRAM [148, 149] makes conventional processor designs impractical in commodity DRAM chips [150–153].

Real-world PIM systems have only recently been manufactured and commercialized. The UPMEM company, for example, introduced the first general-purpose commercial PIM architecture [154–158], which integrates small in-order cores near DRAM memory banks. High-bandwidth memory (HBM)-based HBM-PIM [159, 160] and Acceleration DIMM (AxDIMM) [161] are Samsung's proposals that have been successfully tested via real prototypes. HBM-PIM features *Single Instruction Multiple Data* (*SIMD*) units, which support multiply-add and multiply-accumulate operations, near the banks in HBM layers [162, 163], and it is designed to accelerate neural network inference. AxDIMM is a near-rank solution that places an FPGA fabric on a DDR module to accelerate specific workloads (e.g., recommendation inference). Accelerator-in-Memory (AiM) [164] is a GDDR6-based PIM architecture from SK Hynix with specialized units for multiply-accumulate and activation functions for deep learning. HB-PNM [165] is a 3D-stacked-based PIM architecture from Alibaba, which stacks a layer of LPDDR4 memory and a logic layer with specialized accelerators for recommendation systems.

These five real-world PIM systems have some important common characteristics, as depicted in Figure 1. First, there is a host processor (CPU or GPU), typically with a deep cache hierarchy, which has access to (1) standard main memory, and (2) PIM-enabled memory (i.e., UPMEM DIMMs, HBM-PIM stacks, AxDIMM DIMMs, AiM GDDR6, HB-PNM LPDDR4). Second, the PIM-enabled memory chip contains multiple PIM processing elements (PIM PEs), which have access to memory (either memory banks or ranks) with higher bandwidth and lower latency than the host processor. Third, the PIM processing elements (either general-purpose cores, SIMD units, FPGAs, or specialized processors) run at only a few hundred megahertz, and have a small number of registers and relatively small (or no) cache or scratchpad memory. Fourth, processing elements may not be able to communicate directly with each other (e.g., UPMEM DPUs, HBM-PIM PCUs or

AiM PUs in different chips), and communication between them happens via the host processor. Figure 1 shows a high-level view of such a state-of-the-art processing-in-memory system.

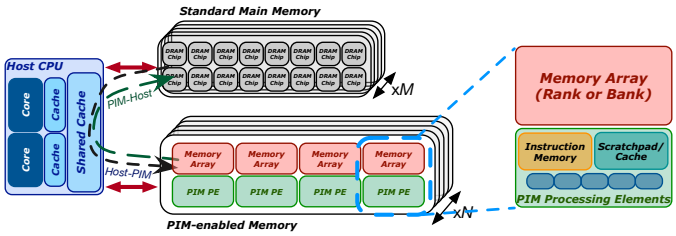

Fig. 1. High-level view of a state-of-the-art processing-in-memory system. The host CPU has access to M standard memory modules and N PIM-enabled memory modules.

Our goal in this work is to quantify the potential of general-purpose PIM architectures for training of machine learning algorithms. To this end, we implement four representative classical machine learning algorithms (linear regression [166, 167], logistic regression [166, 168], decision tree [169], K-means clustering [170]) on a general-purpose memory-centric system containing PIM-enabled memory, specifically the UPMEM PIM architecture [154–158]. We do *not* include training of deep learning algorithms in our study, since GPUs and TPUs have a solid position as the preferred and highly optimized accelerators for deep learning training [90, 171–176].

Our PIM implementations of ML algorithms follow PIM programming recommendations in recent literature [155–157, 177]. We apply several optimizations to overcome the limitations of existing general-purpose PIM architectures (e.g., limited instruction set, relatively simple pipeline, relatively low frequency) and take full advantage of the inherent strengths of PIM (e.g., large memory bandwidth, low memory latency). Table I summarizes characteristics of our ML workloads and their PIM implementations.

TABLE I
MACHINE LEARNING WORKLOADS.

Characteristic		Linear Regression	Logistic Regression	Decision Tree	K-Means
Short name		LIN	LOG	DTR	KME
Learning approach		Supervised			Unsupervised
Application		Regression	Classification		Clustering
Memory access pattern	Sequential	Yes	Yes	Yes	Yes
	Strided	No	No	No	No
	Random	No	No	No	No
Computation pattern	Operations	mul, add	mul, add, exp, div	compare, add	mul, compare, add
	Datatypes	float, int32_t	float, int32_t	float	int16_t, int64_t
Communication/ synchronization	Intra PIM Core	barrier	barrier	barrier, mutex	barrier, mutex
	Inter PIM Core	Yes	Yes	Yes	Yes

We evaluate our PIM implementations in terms of training accuracy, performance, and scaling characteristics on a real memory-centric system with PIM-enabled memory [154, 177, 178]. We run our experiments on a real-world PIM system [154] with 2,524 PIM cores running at 425 MHz, and 158 GB of DRAM memory.[1] Our experimental real system evaluation provides new observations and insights that we summarize in Section II.

We compare our PIM implementations of linear regression, logistic regression, decision tree, and K-means clustering to their state-of-the-art CPU and GPU counterparts. We observe that memory-centric systems with PIM-enabled memory can significantly outperform processor-centric systems for memory-bound ML training workloads, when the operations needed by the ML workloads are natively supported by PIM hardware (or can be replaced by efficient LUT implementations).

Our extended paper [1] contains (1) detailed description of our PIM implementations of ML workloads; (2) comprehensive evaluation and comparisons to state-of-the-art CPU and GPU systems; and (3) more insights about the suitability of ML workloads to the PIM system. We aim to open-source all our PIM implementations of ML training workloads, training datasets, and evaluation scripts.

II. Key Takeaways and Recommendations

In this section, we summarize our key observations, takeaways, and recommendations that stem from our analysis of four ML training workloads on a state-of-the-art general-purpose PIM architecture. There are four subsections that are titled after three widely-accepted facts about near-bank PIM architectures. For each of them, we first state general key observations that are derived from the particular fact. Then, we provide insights derived from our work in the form of key takeaways and recommendations.

A. PIM Cores Are Compute-bound

PIM cores [150, 159, 160, 164] are wimpy processors (operating at relatively low frequency) with high memory bandwidth, especially for streaming memory access patterns [156]. As a result, for real-world workloads, the compute throughput tends to saturate much more frequently than the memory bandwidth, and the pipeline latency hides the memory access latency.

Key Takeaway #1. Even ML training workloads (e.g., linear regression, logarithmic regression, decision tree, K-Means) that are bound by memory access due to their low arithmetic intensity in processor-centric systems (e.g., CPU, GPU) behave as compute-bound when running on PIM cores.

Recommendation #1. Maximize the utilization of PIM cores by keeping their pipeline fully busy. For example, in the UPMEM PIM architecture [150], which has fine-grained multithreaded scalar cores, we recommend to schedule 11 or more PIM threads, which is the minimum number of PIM threads to saturate the pipeline throughput. Figure 2 shows this saturation point for decision tree (`DTR`) and K-Means (`KME`), as the PIM kernel time is flat after 11 PIM threads.

B. PIM Cores Have Limited Instruction Sets

PIM cores [150, 159, 160, 164] have limited instruction sets. As such, they do *not* support natively a large variety of

[1]The UPMEM-based PIM system has up to 2,560 PIM cores and 160 GB of DRAM.

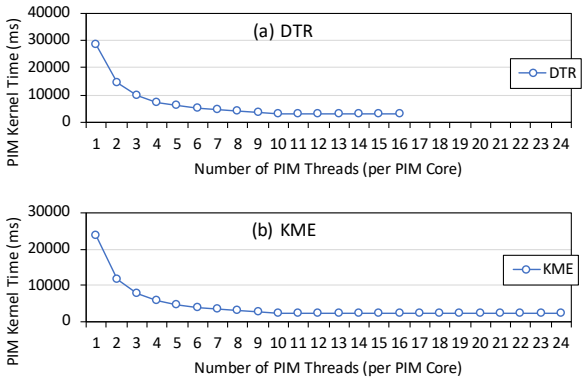

Fig. 2. Execution time (ms) of decision tree (a) on 1-16 PIM threads in 1 PIM core and K-means clustering (b) on 1-24 PIM threads in 1 PIM core.

arithmetic operations and datatypes. For example, the UPMEM PIM architecture [150] does *not* support floating-point operations or 32-bit integer multiplication/division (only emulated by the runtime library) [156]. AiM [164] and HBM-PIM [159, 160] only support multiplication and addition of 16-bit floating-point values.

Key Takeaway #2. Workloads that require arithmetic operations or datatypes that are not natively supported by PIM cores run at low performance due to instruction emulation (e.g., floating-point operations in UPMEM PIM).

Recommendation #2. ML workloads (e.g., linear regression, logistic regression) can employ fixed-point representation if PIM cores do not support floating-point operations (e.g., UPMEM PIM) without sacrificing much accuracy. For example, our fixed-point PIM implementation of linear regression (`LIN`) provides training accuracy of 1.02%, which the training accuracy of the floating-point version is 0.55% (same as the CPU version).

Recommendation #3. Quantization can be used to take advantage of native hardware support, if PIM cores only support limited precision. For example, using hybrid precision after quantizing the training dataset can provide significant performance improvements. Our quantized version of `LIN` is 41% faster than the fixed-point version and an order of magnitude faster than the floating-point version.

Recommendation #4. Programmers (or better compilers) can optimize code at low level to better leverage available native instructions and hardware (e.g, 8-bit integer multiplication in UPMEM DPUs). Our custom 16- and 32-bit integer multiplications significantly improve performance by 25% over compiler-generated code for quantized training datasets.

Key Takeaway #3. Memory-bound ML workloads that require mainly operations natively supported by the PIM architecture (e.g, 32-bit integer addition/subtraction in UPMEM PIM) leverage the large PIM bandwidth, and perform better than their CPU and GPU counterparts. For example, `DTR` only requires comparison and 32-bit integer addition. `KME` employs mainly 16-bit integer arithmetic. As a result, our PIM implementations of `DTR` and `KME` provide better performance than state-of-the-art CPU and GPU versions, as Figure 3 shows.

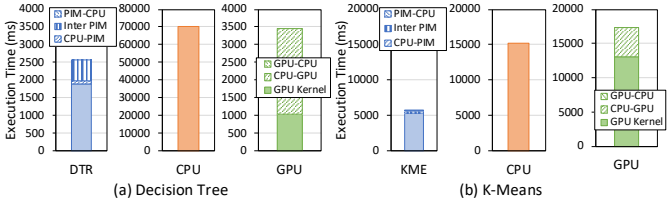

Fig. 3. Execution time (ms) of `DTR` (a) and `KME` (b) on PIM, CPU, and GPU with the Higgs boson segmentation dataset [179]. For the PIM versions (`DTR`, `KME`), the best performing number of PIM cores is 1,024 for `DTR` and 2,524 for `KME`.

C. PIM Cores Have High Memory Bandwidth

Near-bank PIM cores leverage high aggregated memory bandwidth and low latency memory access. As a result, memory accesses are typically cheaper than computation (in particular, 32- or 64-bit integer arithmetic, floating-point arithmetic, transcendental functions) for workloads that are memory bound in processor-centric systems due to the relatively low frequency and simple pipelines of PIM cores [156].

PIM cores exploit memory bandwidth better when they perform streaming memory accesses to the memory banks, especially if memory arrays have large row buffers (e.g., DDR4 memory in UPMEM PIM).

Recommendation #5. Programmers can tradeoff memory accesses for computation in PIM architectures by keeping pre-calculated operation results (e.g., LUTs, memoization) in memory. For example, our use of LUTs for sigmoid calculation in logistic regression (`LOG`) results in a speedup of 53×.

Recommendation #6. For data structures of more than one dimension, programmers can optimize the data layout in a way that memory accesses are in streaming, thus exploiting higher sustained bandwidth. One example of optimized data layout is our PIM implementation of `DTR`.

D. PIM Throughput Scales with Memory Capacity

Since near-bank PIM cores are directly attached to the memory arrays, their number scales at the same pace as the number of memory arrays, banks, and chips in the memory subsystem. Consequently, the overall PIM throughput scales linearly.

Large PIM-enabled memory allows training datasets to remain in memory during the whole training process. This represents an inherent advantage over processor-centric systems (CPU, GPU) where the whole training dataset needs to be moved to the processor in every training iteration.

Key Takeaway #4. Memory-bound ML training workloads, which need large training datasets, benefit from large PIM-enabled memory with many PIM cores. Even if PIM cores need to communicate via the host processor (e.g., in UPMEM PIM), the amount of data movement needed for intermediate results is minimal with respect to the size of the whole training dataset. We observe this with strong scaling and weak scaling characterization of four ML training workloads.

III. Conclusion

Training of machine learning workloads frequently becomes memory-bound in processor-centric systems due to repeated accesses to large training datasets. Memory-centric computing

systems (i.e., with processing-in-memory, PIM, capabilities) can overcome this memory boundedness.

We implement several representative classic machine learning algorithms on a real-world general-purpose PIM architecture with the aim of understanding the potential of memory-centric systems for ML training. We evaluate our PIM implementations on a memory-centric computing system with more than 2500 PIM cores in terms of accuracy, performance, and scaling characteristics, and compare to state-of-the-art implementations for CPU and GPU.

To our knowledge, our work is the first one to evaluate training of machine learning algorithms on a real-world PIM architecture. Outcomes of our work are several key observations, takeaways, and recommendations that are directed at users of machine learning workloads, programmers of PIM architectures, and hardware designers and architects of future memory-centric computing systems. More details about our work are available in [1].

REFERENCES

[1] J. Gómez-Luna, Y. Guo, S. Brocard, J. Legriel, R. Cimadomo, G. F. Oliveira, G. Singh, and O. Mutlu, "An Experimental Evaluation of Machine Learning Training on a Real Processing-in-Memory System," *arXiv preprint arXiv:2207.07886*, 2022.

[2] A. Géron, *Hands-on Machine Learning With Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*, 2019.

[3] E. Alpaydin, *Introduction to Machine Learning*, 2020.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, 2016.

[5] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*, 2018.

[6] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, 2014.

[7] S. Raschka and V. Mirjalili, *Python Machine Learning: Machine Learning and Deep Learning with Python, Scikit-Learn, and TensorFlow 2*, 2019.

[8] O. Mutlu *et al.*, "Processing Data Where It Makes Sense: Enabling In-Memory Computation," *MicPro*, 2019.

[9] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "A Modern Primer on Processing in Memory," *Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann*, 2021, https://arxiv.org/pdf/2012.03112.pdf.

[10] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, "Processing-in-Memory: A Workload-Driven Perspective," *IBM JRD*, 2019.

[11] V. Seshadri and O. Mutlu, "In-DRAM Bulk Bitwise Execution Engine," arXiv:1905.09822 [cs.AR], 2020.

[12] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Enabling Practical Processing in and near Memory for Data-Intensive Computing," in *DAC*, 2019.

[13] H. S. Stone, "A Logic-in-Memory Computer," *IEEE TC*, 1970.

[14] W. H. Kautz, "Cellular Logic-in-Memory Arrays," *IEEE TC*, 1969.

[15] D. E. Shaw, S. J. Stolfo, H. Ibrahim, B. Hillyer, G. Wiederhold, and J. Andrews, "The NON-VON Database Machine: A Brief Overview," *IEEE Database Eng. Bull.*, 1981.

[16] P. M. Kogge, "EXECUBE - A New Architecture for Scaleable MPPs," in *ICPP*, 1994.

[17] M. Gokhale, B. Holmes, and K. Iobst, "Processing in Memory: The Terasys Massively Parallel PIM Array," *IEEE Computer*, 1995.

[18] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A Case for Intelligent RAM," *IEEE Micro*, 1997.

[19] M. Oskin, F. T. Chong, and T. Sherwood, "Active Pages: A Computation Model for Intelligent Memory," in *ISCA*, 1998.

[20] 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 *ICCD*, 1999.

[21] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz, "Smart Memories: A Modular Reconfigurable Architecture," in *ISCA*, 2000.

[22] R. C. Murphy, P. M. Kogge, and A. Rodrigues, "The Characterization of Data Intensive Memory Workloads on Distributed PIM Systems," in *Intelligent Memory Systems*. Springer.

[23] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, I. Kim, and G. Daglikoca, "The Architecture of the DIVA Processing-in-Memory Chip," in *SC*, 2002.

[24] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute Caches," in *HPCA*, 2017.

[25] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural Cache: Bit-serial In-cache Acceleration of Deep Neural Networks," in *ISCA*, 2018.

[26] D. Fujiki, S. Mahlke, and R. Das, "Duality Cache for Data Parallel Acceleration," in *ISCA*, 2019.

[27] M. Kang, M.-S. Keel, N. R. Shanbhag, S. Eilert, and K. Curewitz, "An Energy-Efficient VLSI Architecture for Pattern Recognition via Deep Embedding of Computation in SRAM," in *ICASSP*, 2014.

[28] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.

[29] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM," arXiv:1611.09988 [cs:AR], 2016.

[30] V. Seshadri, K. Hsieh, A. Boroumand, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Fast Bulk Bitwise AND and OR in DRAM," *CAL*, 2015.

[31] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.

[32] S. Angizi and D. Fan, "Graphide: A Graph Processing Accelerator Leveraging In-DRAM-computing," in *GLSVLSI*, 2019.

[33] J. Kim, M. Patel, H. Hassan, and O. Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern DRAM Devices," in *HPCA*, 2018.

[34] J. Kim, M. Patel, H. Hassan, L. Orosa, and O. Mutlu, "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput," in *HPCA*, 2019.

[35] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs," in *MICRO*, 2019.

[36] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.

[37] X. Xin, Y. Zhang, and J. Yang, "ELP2IM: Efficient and Low Power Bitwise Operation Processing in DRAM," in *HPCA*, 2020.

[38] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-Based Reconfigurable In-Situ Accelerator," in *MICRO*, 2017.

[39] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang, "DrAcc: A DRAM Based Accelerator for Accurate CNN Inference," in *DAC*, 2018.

[40] N. Hajinazar, G. F. Oliveira, S. Gregorio, J. D. Ferreira, N. M. Ghiasi, M. Patel, M. Alser, S. Ghose, J. Gómez-Luna, and O. Mutlu, "SIMDRAM: A Framework for Bit-Serial SIMD Processing Using DRAM," in *ASPLOS*, 2021.

[41] S. H. S. Rezaei, M. Modarressi, R. Ausavarungnirun, M. Sadrosadati, O. Mutlu, and M. Daneshtalab, "NoM: Network-on-Memory for Inter-Bank Data Transfer in Highly-Banked Memories," *CAL*, 2020.

[42] Y. Wang, L. Orosa, X. Peng, Y. Guo, S. Ghose, M. Patel, J. S. Kim, J. G. Luna, M. Sadrosadati, N. M. Ghiasi *et al.*, "FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching," in *MICRO*, 2020.

[43] M. F. Ali, A. Jaiswal, and K. Roy, "In-Memory Low-Cost Bit-Serial Addition Using Commodity DRAM Technology," in *TCAS-I*, 2019.

[44] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-Volatile Memories," in *DAC*, 2016.

[45] S. Angizi, Z. He, and D. Fan, "PIMA-Logic: A Novel Processing-in-Memory Architecture for Highly Flexible and Energy-efficient Logic Computation," in *DAC*, 2018.

[46] S. Angizi, A. S. Rakin, and D. Fan, "CMP-PIM: An Energy-efficient Comparator-based Processing-in-Memory Neural Network Accelerator," in *DAC*, 2018.

[47] S. Angizi, J. Sun, W. Zhang, and D. Fan, "AlignS: A Processing-in-Memory Accelerator for DNA Short Read Alignment Leveraging SOT-MRAM," in *DAC*, 2019.

[48] Y. Levy, J. Bruck, Y. Cassuto, E. G. Friedman, A. Kolodny, E. Yaakobi, and S. Kvatinsky, "Logic Operations in Memory Using a Memristive Akers Array," *Microelectronics Journal*, 2014.

[49] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "MAGIC—Memristor-Aided Logic," *IEEE TCAS II: Express Briefs*, 2014.

[50] A. Shafiee, A. Nag, N. Muralimanohar, and et al., "ISAAC: A Convolutional Neural Network Accelerator with In-situ Analog Arithmetic in Crossbars," in *ISCA*, 2016.

[51] S. Kvatinsky, A. Kolodny, U. C. Weiser, and E. G. Friedman, "Memristor-Based IMPLY Logic Design Procedure," in *ICCD*, 2011.

[52] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies," *TVLSI*, 2014.

[53] P.-E. Gaillardon, L. Amaru, A. Siemon, and et al., "The Programmable Logic-in-Memory (PLiM) Computer," in *DATE*, 2016.

[54] D. Bhattacharjee, R. Devadoss, and A. Chattopadhyay, "ReVAMP: ReRAM based VLIW Architecture for In-memory Computing," in *DATE*, 2017.

[55] S. Hamdioui, L. Xie, H. A. D. Nguyen, and et al., "Memristor Based Computation-in-Memory Architecture for Data-intensive Applications," in *DATE*, 2015.

[56] L. Xie, H. A. D. Nguyen, M. Taouil, and et al., "Fast Boolean Logic Papped on Memristor Crossbar," in *ICCD*, 2015.

[57] S. Hamdioui, S. Kvatinsky, and e. a. G. Cauwenberghs, "Memristor for Computing: Myth or Reality?" in *DATE*, 2017.

[58] J. Yu, H. A. D. Nguyen, L. Xie, and et al., "Memristive Devices for Computation-in-Memory," in *DATE*, 2018.

[59] C. Giannoula, N. Vijaykumar, N. Papadopoulou, V. Karakostas, I. Fernandez, J. Gómez-Luna, L. Orosa, N. Koziris, G. Goumas, and O. Mutlu, "SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures," in *HPCA*, 2021.

[60] I. Fernandez, R. Quislant, C. Giannoula, M. Alser, J. Gomez-Luna, E. Gutierrez, O. Plata, and O. Mutlu, "NATSA: A Near-Data Processing Accelerator for Time Series Analysis," in *ICCD*, 2020.

[61] D. S. Cali, G. S. Kalsi, Z. Bingöl, C. Firtina, L. Subramanian, J. S. Kim, R. Ausavarungnirun, M. Alser, J. Gomez-Luna, A. Boroumand *et al.*, "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis," in *MICRO*, 2020.

[62] J. S. Kim, D. Senol, H. Xin, D. Lee, S. Ghose, M. Alser, H. Hassan, O. Ergin, C. Alkan, and O. Mutlu, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies," *BMC Genomics*, 2018.

[63] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.

[64] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.

[65] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.

[66] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, R. Ausavarungnirun, K. Hsieh, N. Hajinazar, K. T. Malladi, H. Zheng, and O. Mutlu, "CoNDA: Efficient Cache Coherence Support for near-Data Accelerators," in *ISCA*, 2019.

[67] G. Singh, J. Gomez-Luna, G. Mariani, G. F. Oliveira, S. Corda, S. Stujik, O. Mutlu, and H. Corporaal, "NAPEL: Near-memory Computing Application Performance Prediction via Ensemble Learning," in *DAC*, 2019.

[68] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, "Chameleon: Versatile and Practical Near-DRAM Acceleration Architecture for Large Memory Systems," in *MICRO*, 2016.

[69] O. O. Babarinsa and S. Idreos, "JAFAR: Near-Data Processing for Databases," in *SIGMOD*, 2015.

[70] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-In-Memory Architecture for Neural Network Computation In ReRAM-Based Main Memory," in *ISCA*, 2016.

[71] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim, "NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules," in *HPCA*, 2015.

[72] M. Gao, G. Ayers, and C. Kozyrakis, "Practical Near-Data Processing for In-Memory Analytics Frameworks," in *PACT*, 2015.

[73] M. Gao and C. Kozyrakis, "HRL: Efficient and Flexible Reconfigurable Logic for Near-Data Processing," in *HPCA*, 2016.

[74] B. Gu, A. S. Yoon, D.-H. Bae, I. Jo, J. Lee, J. Yoon, J.-U. Kang, M. Kwon, C. Yoon, S. Cho, J. Jeong, and D. Chang, "Biscuit: A Framework for Near-Data Processing of Big Data Workloads," in *ISCA*, 2016.

[75] Q. Guo, N. Alachiotis, B. Akin, F. Sadi, G. Xu, T. M. Low, L. Pileggi, J. C. Hoe, and F. Franchetti, "3D-Stacked Memory-Side Acceleration: Accelerator and System Design," in *WoNDP*, 2014.

[76] M. Hashemi, Khubaib, E. Ebrahimi, O. Mutlu, and Y. N. Patt, "Accelerating Dependent Cache Misses with an Enhanced Memory Controller," in *ISCA*, 2016.

[77] M. Hashemi, O. Mutlu, and Y. N. Patt, "Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads," in *MICRO*, 2016.

[78] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O'Conner, N. Vijaykumar, O. Mutlu, and S. Keckler, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *ISCA*, 2016.

[79] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," in *ISCA*, 2016.

[80] G. Kim, N. Chatterjee, M. O'Connor, and K. Hsieh, "Toward Standardized Near-Data Processing with Unrestricted Data Placement for GPUs," in *SC*, 2017.

[81] J. H. Lee, J. Sim, and H. Kim, "BSSync: Processing Near Memory for Machine Learning Workloads with Bounded Staleness Consistency Models," in *PACT*, 2015.

[82] Z. Liu, I. Calciu, M. Herlihy, and O. Mutlu, "Concurrent Data Structures for Near-Memory Computing," in *SPAA*, 2017.

[83] A. Morad, L. Yavits, and R. Ginosar, "GP-SIMD Processing-in-Memory," *ACM TACO*, 2015.

[84] L. Nai, R. Hadidi, J. Sim, H. Kim, P. Kumar, and H. Kim, "GraphPIM: Enabling Instruction-Level PIM Offloading in Graph Computing Frameworks," in *HPCA*, 2017.

[85] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das, "Scheduling Techniques for GPU Architectures with Processing-in-Memory Capabilities," in *PACT*, 2016.

[86] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, "NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads," in *ISPASS*, 2014.

[87] D. P. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski, "TOP-PIM: Throughput-Oriented Programmable Processing in Memory," in *HPDC*, 2014.

[88] 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 *HPEC*, 2013.

[89] B. Akin, F. Franchetti, and J. C. Hoe, "Data Reorganization in Memory Using 3D-Stacked DRAM," in *ISCA*, 2015.

[90] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and Efficient Neural Network Acceleration with 3D Memory," in *ASPLOS*, 2017.

[91] M. Drumond, A. Daglis, N. Mirzadeh, D. Ustiugov, J. Picorel Obando, B. Falsafi, B. Grot, and D. Pnevmatikatos, "The Mondrian Data Engine," in *ISCA*, 2017.

[92] G. Dai, T. Huang, Y. Chi, J. Zhao, G. Sun, Y. Liu, Y. Wang, Y. Xie, and H. Yang, "GraphH: A Processing-in-Memory Architecture for Large-scale Graph Processing," *IEEE TCAD*, 2018.

[93] M. Zhang, Y. Zhuo, C. Wang, M. Gao, Y. Wu, K. Chen, C. Kozyrakis, and X. Qian, "GraphP: Reducing Communication for PIM-based Graph Processing with Efficient Data Partition," in *HPCA*, 2018.

[94] Y. Huang, L. Zheng, P. Yao, J. Zhao, X. Liao, H. Jin, and J. Xue, "A Heterogeneous PIM Hardware-Software Co-Design for Energy-Efficient Graph Processing," in *IPDPS*, 2020.

[95] Y. Zhuo, C. Wang, M. Zhang, R. Wang, D. Niu, Y. Wang, and X. Qian, "GraphQ: Scalable PIM-based Graph Processing," in *MICRO*, 2019.

[96] P. C. Santos, G. F. Oliveira, D. G. Tomé, M. A. Alves, E. C. Almeida, and L. Carro, "Operand Size Reconfiguration for Big Data Processing in Memory," in *DATE*, 2017.

[97] W.-M. Hwu, I. El Hajj, S. G. De Gonzalo, C. Pearson, N. S. Kim, D. Chen, J. Xiong, and Z. Sura, "Rebooting the Data Access Hierarchy of Computing Systems," in *ICRC*, 2017.

[98] M. Besta, R. Kanakagiri, G. Kwasniewski, R. Ausavarungnirun, J. Beránek, K. Kanellopoulos, K. Janda, Z. Vonarburg-Shmaria, L. Gianinazzi, I. Stefan *et al.*, "SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems," in *MICRO*, 2021.

[99] J. D. Ferreira, G. Falcao, J. Gómez-Luna, M. Alser, L. Orosa, M. Sadrosadati, J. S. Kim, G. F. Oliveira, T. Shahroodi, A. Nori, and O. Mutlu, "pLUTo: In-DRAM Lookup Tables to Enable Massively Parallel General-Purpose Computation," *arXiv:2104.07699 [cs.AR]*, 2021.

[100] A. Olgun, M. Patel, A. G. Yağlıkçı, H. Luo, J. S. Kim, F. N. Bostancı, N. Vijaykumar, O. Ergin, and O. Mutlu, "QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAMs," in *ISCA*, 2021.

[101] S. Lloyd and M. Gokhale, "In-memory Data Rearrangement for Irregular, Data-intensive Computing," *Computer*, 2015.

[102] D. G. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, and R. McKenzie, "Computational RAM: Implementing Processors in Memory," *IEEE Design & Test of Computers*, 1999.

[103] L. Zheng, S. Shin, S. Lloyd, M. Gokhale, K. Kim, and S.-M. Kang, "RRAM-based TCAMs for pattern search," in *ISCAS*, 2016.

[104] J. Landgraf, S. Lloyd, and M. Gokhale, "Combining Emulation and Simulation to Evaluate a Near Memory Key/Value Lookup Accelerator," 2021.

[105] A. Rodrigues, M. Gokhale, and G. Voskuilen, "Towards a Scatter-Gather Architecture: Hardware and Software Issues," in *MEMSYS*, 2019.

[106] S. Lloyd and M. Gokhale, "Design Space Exploration of Near Memory Accelerators," in *MEMSYS*, 2018.

[107] S. Lloyd and M. Gokhale, "Near Memory Key/Value Lookup Acceleration," in *MEMSYS*, 2017.

[108] M. Gokhale, S. Lloyd, and C. Hajas, "Near Memory Data Structure Rearrangement," in *MEMSYS*, 2015.

[109] R. Nair, S. F. Antao, C. Bertolli, P. Bose, J. R. Brunheroto, T. Chen, C.-Y. Cher, C. H. Costa, J. Doi, C. Evangelinos *et al.*, "Active Memory Cube: A Processing-in-Memory Architecture for Exascale Systems," *IBM JRD*, 2015.

[110] A. C. Jacob, Z. Sura, T. Chen, C. Bertolli, S. Antao, O. Sallenave, K. O'Brien, H. Jacobson, R. Nair, J. R. Brunheroto *et al.*, "Compiling for the Active Memory Cube," Tech. rep. RC25644 (WAT1612-008). IBM Research Division, Tech. Rep., 2016.

[111] Z. Sura, A. Jacob, T. Chen, B. Rosenburg, O. Sallenave, C. Bertolli, S. Antao, J. Brunheroto, Y. Park, K. O'Brien *et al.*, "Data Access Optimization in a Processing-in-Memory System," in *CF*, 2015.

[112] R. Nair, "Evolution of Memory Architecture," *Proceedings of the IEEE*, 2015.

[113] 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*, 2014.

[114] Y. Xi, B. Gao, J. Tang, A. Chen, M.-F. Chang, X. S. Hu, J. Van Der Spiegel, H. Qian, and H. Wu, "In-Memory Learning With Analog Resistive Switching Memory: A Review and Perspective," *Proceedings of the IEEE*, 2020.

[115] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation," in *ICCD*, 2016.

[116] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, K. Hsieh, K. T. Malladi, H. Zheng, and O. Mutlu, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," *CAL*, 2016.

[117] C. Giannoula, I. Fernandez, J. Gómez-Luna, N. Koziris, G. Goumas, and O. Mutlu, "SparseP: Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Systems," *arXiv preprint arXiv:2201.05072*, 2022.

[118] C. Giannoula, I. Fernandez, J. Gómez-Luna, N. Koziris, G. Goumas, and O. Mutlu, "Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-in-Memory Architectures," in *SIGMETRICS*, 2022.

[119] A. Denzler, R. Bera, N. Hajinazar, G. Singh, G. F. Oliveira, J. Gómez-Luna, and O. Mutlu, "Casper: Accelerating stencil computation using near-cache processing," *arXiv preprint arXiv:2112.14216*, 2021.

[120] A. Boroumand, S. Ghose, G. F. Oliveira, and O. Mutlu, "Polynesia: Enabling Effective Hybrid Transactional/Analytical Databases with Specialized Hardware/Software Co-Design," arXiv:2103.00798 [cs.AR], 2021.

[121] A. Boroumand, S. Ghose, G. F. Oliveira, and O. Mutlu, "Polynesia: Enabling Effective Hybrid Transactional Analytical Databases with Specialized Hardware Software Co-Design," in *ICDE*, 2022.

[122] G. Singh, M. Alser, D. S. Cali, D. Diamantopoulos, J. Gómez-Luna, H. Corporaal, and O. Mutlu, "FPGA-based Near-Memory Acceleration of Modern Data-Intensive Applications," *IEEE Micro*, 2021.

[123] G. Singh, D. Diamantopoulos, J. Gómez-Luna, C. Hagleitner, S. Stuijk, H. Corporaal, and O. Mutlu, "Accelerating Weather Prediction using Near-Memory Reconfigurable Fabric," *ACM TRETS*, 2021.

[124] J. M. Herruzo, I. Fernandez, S. González-Navarro, and O. Plata, "Enabling Fast and Energy-Efficient FM-Index Exact Matching Using Processing-Near-Memory," *The Journal of Supercomputing*, 2021.

[125] L. Yavits, R. Kaplan, and R. Ginosar, "GIRAF: General Purpose In-Storage Resistive Associative Framework," *IEEE TPDS*, 2021.

[126] B. Asgari, R. Hadidi, J. Cao, D. E. Shim, S.-K. Lim, and H. Kim, "FAFNIR: Accelerating Sparse Gathering by Using Efficient Near-Memory Intelligent Reduction," in *HPCA*, 2021.

[127] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks," *arXiv preprint arXiv:2109.14320*, 2021.

[128] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks," in *PACT*, 2021.

[129] A. Boroumand, "Practical Mechanisms for Reducing Processor-Memory Data Movement in Modern Workloads," Ph.D. dissertation, Carnegie Mellon University, 2020.

[130] G. Singh, D. Diamantopoulos, C. Hagleitner, J. Gomez-Luna, S. Stuijk, O. Mutlu, and H. Corporaal, "NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling," in *FPL*, 2020.

[131] V. Seshadri and O. Mutlu, "Simple Operations in Memory to Reduce Data Movement," in *Advances in Computers, Volume 106*, 2017.

[132] S. Diab, A. Nassereldine, M. Alser, J. G. Luna, O. Mutlu, and I. E. Hajj, "High-throughput Pairwise Alignment with the Wavefront Algorithm using Processing-in-Memory," *arXiv preprint arXiv:2204.02085*, 2022.

[133] S. Diab, A. Nassereldine, M. Alser, J. G. Luna, O. Mutlu, and I. E. Hajj, "High-throughput Pairwise Alignment with the Wavefront Algorithm using Processing-in-Memory," in *HICOMB*, 2022.

[134] D. Fujiki, S. Mahlke, and R. Das, "In-Memory Data Parallel Processor," in *ASPLOS*, 2018.

[135] Y. Zha and J. Li, "Hyper-AP: Enhancing Associative Processing Through A Full-Stack Optimization," in *ISCA*, 2020.

[136] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," *IMW*, 2013.

[137] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," *SUPERFRI*, 2014.

[138] H. Ahmed, P. C. Santos, J. P. Lima, R. F. Moura, M. A. Alves, A. C. Beck, and L. Carro, "A Compiler for Automatic Selection of Suitable Processing-in-Memory Instructions," in *DATE*, 2019.

[139] S. Jain, S. Sapatnekar, J.-P. Wang, K. Roy, and A. Raghunathan, "Computing-in-Memory with Spintronics," in *DATE*, 2018.

[140] N. M. Ghiasi, J. Park, H. Mustafa, J. Kim, A. Olgun, A. Gollwitzer, D. S. Cali, C. Firtina, H. Mao, N. A. Alserr *et al.*, "GenStore: A High-Performance and Energy-Efficient In-Storage Computing System for Genome Sequence Analysis," in *ASPLOS*, 2022.

[141] G. F. Oliveira, J. Gómez-Luna, L. Orosa, S. Ghose, N. Vijaykumar, I. Fernandez, M. Sadrosadati, and O. Mutlu, "DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks," *IEEE Access*, 2021.

[142] G. F. Oliveira, J. Gómez-Luna, L. Orosa, S. Ghose, N. Vijaykumar, I. Fernandez, M. Sadrosadati, and O. Mutlu, "DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks," *arXiv:2105.03725 [cs.AR]*, 2021.

[143] S. Cho, H. Choi, E. Park, H. Shin, and S. Yoo, "McDRAM v2: In-Dynamic Random Access Memory Systolic Array Accelerator to Address the Large Model Problem in Deep Neural Networks on the Edge," *IEEE Access*, 2020.

[144] H. Shin, D. Kim, E. Park, S. Park, Y. Park, and S. Yoo, "McDRAM: Low latency and energy-efficient matrix computations in DRAM," *IEEE TCADICS*, 2018.

[145] P. Gu, X. Xie, Y. Ding, G. Chen, W. Zhang, D. Niu, and Y. Xie, "iPIM: Programmable In-Memory Image Processing Accelerator using Near-Bank Architecture," in *ISCA*, 2020.

[146] D. Lavenier, R. Cimadomo, and R. Jodin, "Variant Calling Parallelization on Processor-in-Memory Architecture," in *BIBM*, 2020.

[147] V. Zois, D. Gupta, V. J. Tsotras, W. A. Najjar, and J.-F. Roy, "Massively Parallel Skyline Computation for Processing-in-Memory Architectures," in *PACT*, 2018.

[148] D. Weber, A. Thies, U. Kahler, M. Lepper, and R. Schutz, "Current and Future Challenges of DRAM Metallization," in *IITC*, 2005.

[149] Y. Peng, B. W. Ku, Y. Park, K.-I. Park, S.-J. Jang, J. S. Choi, and S. K. Lim, "Design, Packaging, and Architectural Policy Co-optimization for DC Power Integrity in 3D DRAM," in *DAC*, 2015.

[150] F. Devaux, "The True Processing In Memory Accelerator," in *Hot Chips*, 2019.

[151] M. Yuffe, E. Knoll, M. Mehalel, J. Shor, and T. Kurts, "A Fully Integrated Multi-CPU, GPU and Memory Controller 32nm processor," in *ISSCC*, 2011.

[152] R. Christy, S. Riches, S. Kottekkat, P. Gopinath, K. Sawant, A. Kona, and R. Harrison, "8.3 A 3GHz ARM Neoverse N1 CPU in 7nm FinFET for Infrastructure Applications," in *ISSCC*, 2020.

[153] T. Singh, S. Rangarajan, D. John, C. Henrion, S. Southard, H. McIntyre, A. Novak, S. Kosonocky, R. Jotwani, A. Schaefer, E. Chang, J. Bell, and M. Co, "3.2 Zen: A Next-generation High-performance x86 Core," in *ISSCC*, 2017.

[154] UPMEM, "UPMEM Website," https://www.upmem.com, 2020.

[155] UPMEM, "Introduction to UPMEM PIM. Processing-in-memory (PIM) on DRAM Accelerator (White Paper)," 2018.

[156] J. Gómez-Luna, I. E. Hajj, I. Fernández, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture," arXiv:2105.03814 [cs.AR], 2021.

[157] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System," *IEEE Access*, 2022.

[158] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-In-Memory Hardware," in *IGSC*, 2021.

[159] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, H.-S. Yu, H. Lee, S. Y. Kim *et al.*, "25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2 TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications," in *ISSCC*, 2021.

[160] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin *et al.*, "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology: Industrial Product," in *ISCA*, 2021.

[161] L. Ke, X. Zhang, J. So, J.-G. Lee, S.-H. Kang, S. Lee, S. Han, Y. Cho, J. H. Kim, Y. Kwon *et al.*, "Near-Memory Processing in Action: Accelerating Personalized Recommendation with AxDIMM," *IEEE Micro*, 2021.

[162] JEDEC, "High Bandwidth Memory (HBM) DRAM," Standard No. JESD235, 2013.

[163] D. Lee, S. Ghose, G. Pekhimenko, S. Khan, and O. Mutlu, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," *TACO*, 2016.

[164] S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim, J. Jeon, N. Kim, Y. Kwon, K. Vladimir, W. Shin, J. Won, M. Lee, H. Joo, H. Choi, J. Lee, D. Ko, Y. Jun, K. Cho, I. Kim, C. Song, C. Jeong, D. Kwon, J. Jang, I. Park, J. Chun, and J. Cho, "A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications," in *ISSCC*, 2022.

[165] D. Niu, S. Li, Y. Wang, W. Han, Z. Zhang, Y. Guan, T. Guan, F. Sun, F. Xue, L. Duan *et al.*, "184QPS/W 64Mb/mm2 3D Logic-to-DRAM Hybrid Bonding with Process-Near-Memory Engine for Recommendation System," in *ISSCC*, 2022.

[166] D. A. Freedman, *Statistical Models: Theory and Practice*, 2009.

[167] X. Yan and X. Su, *Linear Regression Analysis: Theory and Computing*, 2009.

[168] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*, 2013.

[169] S. Suthaharan, "Decision Tree Learning," in *Machine Learning Models and Algorithms for Big Data Classification*, 2016.

[170] S. P. Lloyd, "Least Squares Quantization in PCM," *IEEE Transactions on Information Theory*, 1982.

[171] D. B. Kirk, W.-M. W. Hwu, and B. Ginsburg, *Programming Massively Parallel Processors, 3rd Edition, Chapter 16 - Application Case Study: Machine Learning*. Morgan Kaufmann, 2017.

[172] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient Primitives for Deep Learning," *arXiv preprint arXiv:1410.0759*, 2014.

[173] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A System for Large-scale Machine Learning," in *OSDI*, 2016.

[174] Run:AI, "Best GPU for Deep Learning," https://www.run.ai/guides/gpu-deep-learning/best-gpu-for-deep-learning/, 2021.

[175] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *ISCA*, 2017.

[176] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma *et al.*, "Ten Lessons from Three Generations Shaped Google's TPUv4i: Industrial Product," in *ISCA*, 2021.

[177] UPMEM, "UPMEM User Manual. Version 2021.3.0," 2021.

[178] UPMEM, "UPMEM Software Development Kit (SDK)." https://sdk.upmem.com, 2021.

[179] D. Dua and C. Graff, "UCI Machine Learning Repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml