The Road to Widely Deploying Processing-in-Memory: Challenges and Opportunities

Saugata Ghose
University of Illinois Urbana-Champaign

Processing-in-memory (PIM) refers to a computing paradigm where some or all of the computation for an application is moved closer to where the data resides (e.g., in main memory). While PIM has been the subject of ongoing research since the 1970s [8, 11, 17, 19, 26, 28, 29, 33], it has experienced a resurgence in the last decade due to (1) the pressing need to reduce the energy and latency overheads associated with data movement between the CPU and memory in conventional systems [6, 18], and (2) recent innovations in memory technologies that can enable PIM integration (e.g., [13, 14, 15, 16, 20, 21, 24, 31]). Recently-released products and prototypes, ranging from programmable near-memory processing units [7, 36] to custom near-bank accelerators for machine learning [22, 23, 30] and analog compute support within memory arrays [9, 27], have demonstrated the viability of manufacturing PIM architectures.

Unfortunately, despite the existence of commercial products, PIM has yet to be widely integrated in systems. In this talk, we discuss roadblocks that remain to achieving the widespread adoption of PIM, and highlight some of our recent and ongoing efforts to addressing these roadblocks. The roadblocks that we discuss require the analysis of and co-design across multiple layers of the hardware/software stack. Even with the rapid growth of PIM research in the last decade, the overwhelming majority of these works take a narrow view of the stack, and predominantly focus on either device-level research or proposing new hardware architectures for PIM. We focus on two broad categories of roadblocks that have been neglected by much of the prior work on PIM: (1) incorporating device-level limitations on functionality and manufacturability into architecture-level design decisions, and (2) providing programmer tools and systems software to harness the potential improvements offered by PIM hardware.

First, there is a pressing need for cooperation across the different hardware layers of the stack in order to ensure the viability of PIM architectures. This need is particularly important for *processing-using-memory* (PUM) [10, 32], where electrical interactions between multiple memory cells are used to perform logic and/or arithmetic operations without the need for conventional CMOS-based ALUs. One example of PUM is the MAGIC logic family [20], which can perform Boolean logic primitives (e.g., NOR) using a pair of redox-based resistive RAM (ReRAM) [37] cells. Such Boolean primitives can be used to perform bit-serial operations [2], which perform more complex operations one bit at a time (e.g., a ripple carry addition), but can significantly increase the operation latency compared to a multi-bit CMOS-based ALU. While several PUM architectures propose to overcome the high latencies of bit-serial computation by performing column-wide Boolean primitives, we find that fundamental device- and circuit-level limitations (e.g., the current carrying capacity of a metal wire) prevent PUM architectures from making use of ReRAM arrays larger than 200×200 cells [34]. Working together at the device, circuit, and architecture levels, we overcome several limitations at each level in our RACER architecture for PUM [34, 35]. To date, our work on RACER has resulted in fully-implemented circuits for PUM control flow, improved logic families that are compatible with typical ReRAM devices that can be manufactured today, efficient peripheral circuitry for small memory arrays, and a new execution model that exploits pipelining at the bit granularity. Without further cross-stack hardware projects, a number of other hurdles will continue to remain for the commercial production of PIM architectures.

Second, without further effort on the software stack, it will be difficult to exploit most of the large benefits that PIM architectures promise to provide. Today, this software stack is largely missing, and as a result (1) few PIM works discuss programmer interfaces to the novel hardware being proposed; and (2) there is little support to integrate PIM hardware with existing operating systems and runtimes. On the programming front, some works have proposed frameworks (e.g., [1, 12]) or APIs [25] to implement PIM instructions, while others have exposed PIM functionality as vector instructions [5, 34]. However, a general-purpose compiler for PIM, where the programmer needs little to no knowledge of the underlying hardware, has yet to be developed. Two key challenges to the development of such a compiler are (1) the need for automated approaches to identify opportunities to offload computation to memory (some limited heuristics have been developed [3, 10], but these are not yet fully automated), and (2) the lack of automated data placement strategies to map computation to the near-memory logic units or the memory arrays that contain the data necessary for an operation. On the systems front, programmers have become accustomed to the existence of several support mechanisms that significantly ease multiprocess execution, such as virtual memory, runtime thread scheduling, and cache coherence. There has been some work on coherence mechanisms for PIM [4, 38], but little work exists on general-purpose PIM mechanisms for virtual memory

or thread scheduling. There is a need for the community to start exploring these systems integration issues in order to enable general-purpose use cases for PIM. If the community does not provide support for programmers to use PIM without having to give up existing comforts and programming models, PIM hardware may not be able to overcome the technology adoption chasm and may experience a premature demise.

We hope to inspire a new wave of research that concentrates on the cross-stack issues that we highlight. We believe that both of these sets of roadblocks require interdisciplinary co-design in order to achieve efficient and practical solutions. Such solutions will likely be important enablers to unlock the large benefits promised by PIM architectures.

ACKNOWLEDGMENTS

We thank our many collaborators for their contributions to the works discussed above, including (but not limited to) Minh S. Q. Truong, Amirali Boroumand, Damla Senol Cali, Ryan Wong, Yiqiu Sun, Liting Shen, Alexander Glass, Eric Chen, Deanyone Su, Alison Hoffmann, Ziyi Zuo, L. Richard Carley, James A. Bain, Geraldo F. Oliveira, Nastaran Hajinazar, Juan Gómez-Luna, Jeremie S. Kim, Can Alkan, Sreenivas Subramoney, Gurpreet S. Kalsi, Eric Shiu, Parthasarathy Ranganathan, and Onur Mutlu. The RACER work was funded in part by a seed grant from the Wilton E. Scott Institute for Energy Innovation, and by the Data Storage Systems Center at Carnegie Mellon University. Minh S. Q. Truong is supported by an Apple Ph.D. Fellowship in Integrated Systems.

REFERENCES

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

[2] K. E. Batcher, "Bit-Serial Parallel Processing Systems," *TC*, May 1982.

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

[4] 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.

[5] H. Caminal, K. Yang, S. Srinivasa, A. K. Ramanathan, K. Al-Hawaj, T. Wu, V. Narayanan, C. Batten, and J. F. Martínez, "CAPE: A Content-Addressable Processing Engine," in *HPCA*, 2021.

[6] W. J. Dally, "Challenges for Future Computing Systems," keynote talk at HiPEAC, 2015.

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

[8] 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.

[9] L. Fick, S. Skrzyniarz, M. Parikh, M. B. Henry, and D. Fick, "Analog Matrix Processor for Edge AI Real-Time Video Analytics," in *ISSCC*, 2022.

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

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

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

[13] Hybrid Memory Cube Consortium, "Hybrid Memory Cube Specification 2.1," Oct. 2015.

[14] JEDEC Solid State Technology Assn., *JESD235C: High Bandwidth Memory (HBM) DRAM*, Jan. 2020.

[15] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, "A 28 nm Configurable Memory (TCAM/BCAM/SRAM) Using Push-Rule 6T Bit Cell Enabling Logic-in-Memory," in *JSSC*, 2016.

[16] 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.

[17] 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.

[18] G. Kestor, R. Gioiosa, D. J. Kerbyson, and A. Hoisie, "Quantifying the Cost of Data Movement in Scientific Applications," in *IISWC*, 2013.

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

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

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

[22] Y. C. Kwon, S. H. Lee, J. Lee, S. H. Kwon, J. M. Ryu, J. P. Son, S. O, H. S. Yu, H. Lee, S. Y. Kim, Y. Cho, J. G. Kim, J. Choi, H. S. Shin, J. Kim, B. S. Phuah, H. M. Kim, M. J. Song, A. Choi, D. K. Y. Kim, E. B. Kim, D. Wang, S. Kang, Y. Ro, S. Seo, J. H. Song, J. Youn, K. Sohn, and N. S. Kim, "A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications," in *ISSCC*, 2021.

[23] S. Lee, S.-H. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, S. O, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology," in *ISCA*, 2021.

[24] B. Li, L. Xia, P. Gu, Y. Wang, and H. Yang, "Merging the Interface: Power, Area, and Accuracy Co-Optimization for RRAM Crossbar-Based Mixed-Signal Computing System," in *DAC*, 2015.

[25] E. Lockerman, A. Feldmann, M. Bakhshalipour, A. Stanescu, S. Gupta, D. Sanchez, and N. Beckmann, "Livia: Data-Centric Computing Throughout the Memory Hierarchy," in *ASPLOS*, 2020.

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

[27] Mythic, Inc., "M1076 Analog Matrix Processor," https://mythic.ai/products/m1076-analog-matrix-processor/.

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

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

[30] Samsung Electronics Co., Ltd., "HBM Processing in Memory," https://www.samsung.com/semiconductor/solutions/technology/hbm-processing-in-memory/.

[31] 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.

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

[33] H. S. Stone, "A Logic-in-Memory Computer," *IEEE Trans. Comput.*, Jan. 1970.

[34] M. S. Q. Truong, E. Chen, D. Su, A. Glass, L. Shen, L. R. Carley, J. A. Bain, and S. Ghose, "RACER: Bit-Pipelined Processing Using Resistive Memory," in *MICRO*, 2021.

[35] M. S. Q. Truong, L. Shen, A. Glass, A. Hoffmann, L. R. Carley, J. A. Bain, and S. Ghose, "Adapting the RACER Architecture to Integrate Improved In-ReRAM Logic Primitives," *JETCAS*, Jun. 2022.

[36] UPMEM SAS, "Technology," https://www.upmem.com/technology/.

[37] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal-Oxide RRAM," *Proc. IEEE*, Jun. 2012.

[38] S. Xu, X. Chen, Y. Wang, Y. Han, and X. Li, "CuckooPIM: An Efficient and Less-Blocking Coherence Mechanism for Processing-in-Memory Systems," in *ASPDAC*, 2019.