Expert Opinion

Memristive Accelerators for Dense and Sparse Linear Algebra: From Machine Learning to High-Performance Scientific Computing

Engin Ipek
University of Rochester

Computational models of complex systems are essential to our ability to interpret and understand the world. These models take many forms, including neural networks to interpret and classify vast amounts of real world data,[1] and high fidelity simulations of dynamic systems using partial differential equations.[2] Although these models differ, they utilize many of the same basic computational kernels from linear algebra. Therefore, improvements to the underlying linear algebra kernels can have a significant impact on a wide variety of workloads, including machine learning, scientific computing,[2] graph analytics,[3] and emerging augmented and virtual reality (AR/VR) applications.[4]

One major obstacle to energy efficient linear algebra is data movement. Over the past decade, the energy cost of moving operands for a double precision floating-point operation across an IC has risen to over 10× the energy of the operation itself.[5] At the same time, to meet the rising demand for memory bandwidth, the transmission rate of the off-chip interconnects has increased faster than energy per bit has fallen, resulting in higher peak power consumption.[6] Based on these trends, the U.S. Department of Energy has labeled energy efficient data movement as one of the ten greatest challenges to exascale computing.[7] The high cost of data movement is especially acute when dealing with large working sets, typical of linear algebra workloads for scientific computing and machine learning. The AlexNet deep neural network, for instance, has 240 MB of weights, well beyond typical sizes of last level caches, necessitating numerous off-chip accesses.[8] (Even compressed versions require 6.9 MB.)[9] In scientific computing, the workloads are even larger, and are often distributed across multiple clusters in a supercomputer.

To ameliorate these problems, recent work has focused on a class of *in situ* accelerators that leverage the availability of dense, CMOS-compatible resistive memories.[10–14] *In situ* acceleration exploits the analog properties of a resistive memory array, often a crossbar, to perform computation in the analog domain such that the result of a dot product can be read from the bitlines of an array directly. Recent work has focused on *in situ* matrix-vector multiplication (MVM); however, other accelerators for applications including associative computing with content addressable memories have also been proposed.[15]

In situ acceleration differs from the processing in memory (PIM) concept that has been widely explored as a potential solution to the data movement challenge.[16] PIM-based solutions

Digital Object Identifier 10.1109/MM.2018.2885498
Date of current version 21 February 2019.

 Published by the IEEE Computer Society

colocate processing and memory to reduce the distance that data must travel. However, PIM still follows the same basic framework as conventional architectures, where operands are read individually from memory and processed, thereby neglecting a potentially significant source of data movement within the memory array. *In situ* acceleration, by contrast, performs computation within the array in the analog domain, and reads out the result of the computation with an analog-to-digital converter (ADC).

In situ acceleration exploits the analog properties of a resistive memory array, often a crossbar, to perform computation in the analog domain such that the result of a dot product can be read from the bitlines of an array directly.

IN SITU MVM ACCELERATORS

In an *in situ* MVM accelerator, each resistive memory element in a memory array is programmed inversely proportional to a corresponding matrix coefficient. Although many emerging memory technologies have significant write latencies, the overhead can be amortized over many computations that use the same matrix. Once the values have been programmed into the array, a voltage proportional to a vector coefficient is applied to each wordline of the memory array using a digital-to-analog converter (DAC), energizing the entire array simultaneously. Within the analog domain, the currents through each element in a column sum together, and the resulting current is quantized through an ADC.

The explanation above assumes high precision ADCs and DACs, and cells that can be programmed reliably to tens of bits of precision, all of which are costly at best, and potentially infeasible. To remedy this limitation, prior work on *in situ* MVM leverages a technique called *bit slicing*.[10] The matrix is split into bit planes where each plane contains the data residing at a particular bit position of every matrix coefficient. Each *in situ* operation produces the product between a matrix bit slice and a vector bit slice, the results of which are combined outside the array using a shift and add reduction network. As a result, the total number of crossbar operations required for a computation is equal to the product of the number of matrix and vector bit slices.

CHALLENGES

The first architecture papers on *in situ* MVM acceleration[10–13] focused on machine learning and optimization problems since the requirements of these applications are well suited to *in situ* acceleration. The accelerators rely on three properties of the workloads they accelerate: 1) the inherent error tolerance of the applications; 2) the ability to perform computation using narrow, fixed-point operands; and 3) the dense structure of the weight matrices involved in the computations. Although machine learning and combinatorial optimization are important, the utility of *in situ* acceleration would be limited if these were the only possible use cases. To enable a wider range of applications, including scientific computing and graph analytics, requires three additional capabilities: reliable computation, floating-point support, and efficient handling of sparse matrices.

Reliability

Error accumulation in the analog domain has been a consistent issue with analog computing since the noise margins are much smaller than in a conventional digital system. These smaller noise margins make parasitic noise sources within the memory arrays an even more pressing problem than in conventional memory array design. Additionally, Hamming Codes and other stronger forms of ECC typically used in von-Neumann systems cannot directly be applied to *in situ* computation. Hamming Codes require the values to be read, then corrected, and finally used for computation; in *in situ* computation, however, a dot-product is first computed within the memory array, then read, and only corrected at the end. Therefore, ECC for *in situ* computation must be preserved under addition operations.

There have been several proposals to improve the reliability of computation using *in situ* accelerators. Hu *et al.*[17] demonstrated that by using the continuous programmability of memristive devices, each array element can be programmed to compensate for the specific parasitic effects it is subject to. To support error correction, Feinberg *et al.*[18] propose applying AN codes, a type of arithmetic code where operands are

encoded through multiplication by a carefully selected integer. This technique preserves addition through the distributive property, with detection and correction capabilities respectively implemented as division and remainder operations. Further improvements to AN codes can be achieved by exploiting the strong state dependence of transient errors in resistive memory devices, and by allocating the correction capabilities to minimize the probability of errors.

Further improvements to AN codes can be achieved by exploiting the strong state dependence of transient errors in resistive memory devices, and by allocating the correction capabilities to minimize the probability of errors.

Floating Point

In situ computation is fundamentally based on a fixed-point representation, which creates significant challenges for efficiently performing floating-point calculations. Because values must be aligned prior to addition in fixed-point emulation of floating-point, the mantissa values must be padded based on the difference between the exponents of the two values. This padding can significantly increase the size of the operands at the cost of storage, energy, and latency. In the worst case, double precision floating-point values require 2100 bits (2046 pad bits, a 53 bit mantissa, and a sign bit), and by extension 2100^2 crossbar operations, assuming single bit DACs and memory cells. To mitigate these prohibitive overheads, Feinberg *et al.*[19] propose several techniques. These techniques leverage the properties of floating-point matrices, including exponent range locality, to reduce the overheads to an acceptable level. Exponent range locality is the observation that although the worst-case exponent range requires 2100 bits, the range of exponents within a matrix, or a portion of a matrix, can be captured with significantly fewer bits.

Sparse Matrices

Dense matrices (i.e., matrices with very few zero coefficients) are the ideal case for *in situ* computation. In analog MVM with a crossbar, the entire memory array is charged regardless of the contents of the matrix, consuming a largely fixed amount of power regardless of the matrix density. Therefore, when operating on sparse matrices with a large number of zeros, the efficiency of *in situ* computation is greatly diminished. Conventional techniques for handling sparse matrices rely heavily on indirection, encoding nonzeros as <coordinate, value> tuples to allow zeros to be skipped; however, there is no analog to this capability in *in situ* systems.

Blocking, where the matrix is split into contiguous blocks and blocks of all-zeros are discarded, is a natural analog to mapping the matrix to a set of memory arrays. However, the choice of block size is essential to achieving efficient blocking. GraphR[14] selects smaller arrays of 8×8, using the arrays as a form of MVM functional unit, and streaming a sparse matrix representation of a graph through the arrays. Alternatively, Feinberg *et al.*[19] propose using a set of larger arrays with different sizes to capture the dense blocks within the matrix, processing some of the remaining values that are ill-suited to blocking with a conventional von-Neumann processor.

CONCLUSION

Initial research shows that *in situ* accelerators can be architected to meet the requirements of applications beyond machine learning while still preserving the performance and energy benefits of *in situ* acceleration. However, challenges remain to improve the efficiency of these techniques. Although memristive accelerators significantly reduce data movement, coordinating computation, and marshaling data within a large accelerator, or between systems of multiple accelerators, remains largely unexplored. Additionally, many proposed *in situ* accelerators rely on the matrix being written infrequently such that writes can be amortized over a large number of MVM operations, limiting the applicability to workloads with infrequent matrix updates. If these challenges are addressed, *in situ* memristive computation can provide benefits to an enormous number of applications with linear algebra at their core.

REFERENCES

1. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 39, pp. 436–444, May 2015.

2. Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA, USA: SIAM, 2003.
3. "GraphMat: High performance graph analytics made productive," *Proc. VLDB Endow.*, vol. 8, no. 10, pp. 1214–1225, Jul. 2015.
4. R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G^2o: A general framework for graph optimization," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 3607–3613.
5. S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, Sep./Oct. 2011.
6. D. Lee, M. O'Connor, and N. Chatterjee, "Reducing data transfer energy by exploiting similarity within a data transaction," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2018, pp. 40–51.
7. "Top ten exascale research challenges," DOE Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee Report, 2014.
8. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
9. S. Han, H. Mao, and W. J. Dally, "Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–14.
10. M. N. Bojnordi and E. Ipek, "Memristive Boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2016, pp. 1–13.
11. A. Shafiee *et al.,* "ISAAC: A convolutional neural network accelerator with in situ analog arithmetic in crossbars," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit.*, 2016, pp. 14–26.
12. P. Chi *et al.,* "PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit.*, 2016, pp. 27–39.
13. L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2017, pp. 541–552.
14. L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "GraphR: Accelerating graph processing using ReRAM," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2018, pp. 531–543.
15. Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, "AC-DIMM: Associative computing with STT-MRAM," in *Proc. 40th Annu. Int. Symp. Comput. Archit.*, 2013, pp. 189–200.
16. M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: The Terasys massively parallel PIM array," *Computer*, vol. 28, no. 4, pp. 23–31, Apr. 1995.
17. M. Hu *et al.*, "Dot-product engine for neuromorphic computing: programming 1T1M crossbar to accelerate matrix-vector multiplication," in *Proc. 53rd Annu. Design Automat. Conf.*, 2016, pp. 1–6.
18. B. Feinberg, S. Wang, and E. Ipek, "Making memristive neural network accelerators reliable," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2018, pp. 52–65.
19. B. Feinberg, U. K. R. Vengalam, N. Whitehair, S. Wang, and E. Ipek, "Enabling scientific computing on memristive accelerators," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit.*, 2018, pp. 367–382.

Engin Ipek is an Associate Professor of Electrical and Computer Engineering at the University of Rochester, Rochester, NY, USA. Contact him at ipek@cs.rochester.edu.

This article originally appeared in IEEE Micro, *vol. 39, no. 1, 2019.*