Novel In-Memory Matrix-Matrix Multiplication with Resistive Cross-Point Arrays

Yan Liao, Huaqiang Wu*, Weier Wan#, Wenqiang Zhang, Bin Gao, H.-S. Philip Wong#, and He Qian
Institute of Microelectronics, Tsinghua University, Beijing, China. #Stanford University, USA
*E-mail: wuhq@tsinghua.edu.cn

Abstract

Resistive cross-point array can be used to implement vector-matrix multiplication in analog fashion. However, the output is in the form of analog current, and thus requires A/D conversion prior to digital storage. This paper develops and demonstrates a novel in-memory matrix-matrix multiplication method (M2M) that can compute and store the result directly inside the memory itself without requiring A/D conversion. Compared with the conventional approach, M2M provides $> 10\times$ improvement in energy and area efficiency, and another 2 orders improvement when matrices are low-rank and sparse.

Introduction

Resistive cross-point array can implement vector-matrix multiplication (VMM) by mapping the matrix to resistance state of devices [1]. However, several challenges limit its applications as general-purpose matrix-matrix multiplication accelerator: 1) cross-point array output needs to undergo expensive A/D conversion; 2) the computed results are not directly stored in memory array.

This paper presents a novel in-memory matrix-matrix multiplication method (M2M) using resistive cross-point arrays. Multiplications are implemented in resistive cross-point array and results are directly stored in the same array without requiring A/D conversion. M2M is more energy and area efficient than conventional RRAM-based VMM for many matrix-matrix multiplication applications.

Novel In-Memory Matrix-Matrix Multiplication Method

Fig. 1(a) shows the procedures of the conventional RRAM-based vector-matrix multiplication method. Its time complexity for calculating $A \times B = C$ ($A \in R^{n\times k}$, $B \in R^{k\times m}$) is $O(m)$. A different but equivalent way to calculate the multiplication can be represented as follow:

$$A \times B = \begin{pmatrix} a_{11} \\ \vdots \\ a_{m1} \end{pmatrix} \times (b_{11} \dots b_{1n}) + \cdots + \begin{pmatrix} a_{1k} \\ \vdots \\ a_{mk} \end{pmatrix} \times (b_{k1} \dots b_{kn})$$

One column of A is multiplied with one row of B to obtain one constituent matrix for C. The final result is the sum of constituent matrices (Fig. 1(b) and 1(c)) after k cycles. Resistive cross-point array that allow incremental analog resistance updates provides an efficient way to implement such computation. By applying programming pulses on the rows and columns of a cross-point array, RRAM devices (Fig. 2(a)) act simultaneously as AND gates (binary multiplier), accumulators and storage [2]. Fig. 3 demonstrates the parallel calculation scheme for a given example. Its time complexity of carrying out $A \times B = C$ ($A \in R^{n\times k}$, $B \in R^{k\times m}$) is O(k).

Demonstration on RRAM Array

Fig. 2 shows the microphotograph of the chip used in measurement. The device-to-device and cycle-to-cycle variations of the devices are shown in Fig. 4 and Fig. 5. A 3×3 cross-point array is selected in the chip to carry out multiplication for different input matrix patterns. Average analog behavior for the 3×3 array is shown in Fig. 6. The measurement results for one example are demonstrated in Fig. 7. Measurements were performed for 10^2 cycles for different input patterns. We found that the measured average error (Fig. 8) decreases significantly when the dimension of matrix grows, resulting from programming variability being averaged out by more operating cycles.

Accuracy, Sparsity, and Scalability

A compact RRAM model [3] that includes device-to-device and cycle-to-cycle variations is built for simulation. Fig. 9 shows that the average error is much smaller for larger matrices, which is consistent to the measurement results. For devices with larger variations, lower voltage pulses can be used to slow down the device resistance change per applied pulse such that the variation can be averaged out by more pulses within each quantized level (Fig. 10). Comparing two sets of input matrices with different sparsity (percentage of zero), Fig. 11 shows that the 90% sparse matrices consume $25\times$ lower energy but have $2\times$ higher average error than the 50% sparse matrices. Lower programming voltages can be used to recover the accuracy of sparse matrices, but will increases energy consumption by around $2\times$ due to more operating cycles. Different from conventional cross-point based VMM, the processing time and energy of in-memory M2M scale proportionally with the rank of the matrices (Fig. 12), making it suitable for multiplication ($A \times B = C$, $A \in R^{n\times k}$, $B \in R^{k\times m}$) whose dimension $k \ll m$. Such property is found for matrix low-rank-approximation (LRA) [4], which is commonly used for data compression and recovery. Without precision loss, M2M can achieve $5\times$ improvement in latency and energy for matrices whose rank ratio (k/m) are 0.1.

Performance comparison with RRAM-based VMM

Fig. 13 shows the system configurations of VMM and M2M that are used to estimate the latency, energy, and area. For a fair comparison, in both configurations, the final results are stored in cross-point array. The dimensions of the two input matrices are 128×32 and 32×128 respectively. Matrices with such k/m ratio can be commonly found in applications such as the recovery of LRA. The VMM system would need 128 operating cycles while the proposed in-memory M2M requires 32 cycles. Compared to VMM, in-memory M2M system could achieve savings of 75% in time, 43% in energy, and 70% in area. Besides the difference in the number of operating cycles, the benefits of M2M mainly come from eliminating A/D data conversion [5] and explicit data transfer between the array used for computation and the array used for result storage. Moreover, the energy and area benefits of M2M would be more significant for higher RRAM resistance (Fig. 15), and when matrices are sparse and low-rank.

Conclusion

In this paper, we propose and demonstrate a novel in-memory matrix-matrix multiplication approach that can store computed results directly inside a cross-point memory array without requiring A/D conversion. It enables highly efficient computing kernel that can be widely employed in applications such as data compression and image processing.

Acknowledgements: This work is supported in part by the MOST of China (2016YFA0201801), the ICFC, NSFC (61674089), NSF (1317470), and Stanford SystemX Alliance.

References: [1] M. Hu, DAC, 2016. [2] H. Li, VLSI, 2016. [2] H. Wu, IEDM, 2017. [4] Z. Zhang, Linear Algebra and its Applications, 2003. [5] B. Murmann, "ADC Survey", 2015.

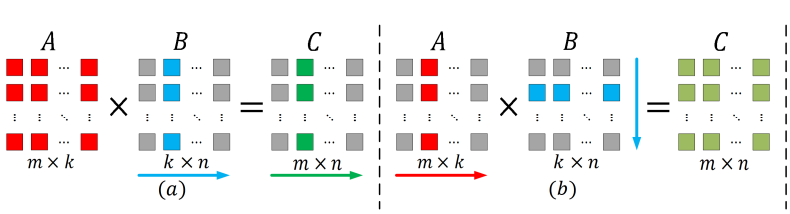

Fig. 1 (a) Procedures of conventional cross-point based vector-matrix multiplication. A is multiplied with each column of B to obtain each column of C (b) Procedures of in-memory matrix-matrix multiplication. One column of A is multiplied with one row of B to obtain one constituent matrix of C (c) During ith cycle, the multiplication of ith column vector in matrix A and ith row vector in Matrix B is implemented, then the results are accumulated with previous results. C_i is the results after i cycles.

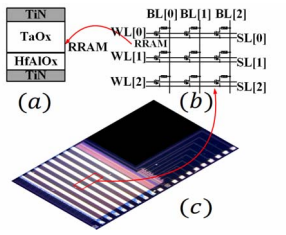

Fig.2 (a) Structure of RRAM device. (b) Circuit diagram of a 3×3 cross-point array. (c) Microphotograph of a 1k-bit 1T1R RRAM array.

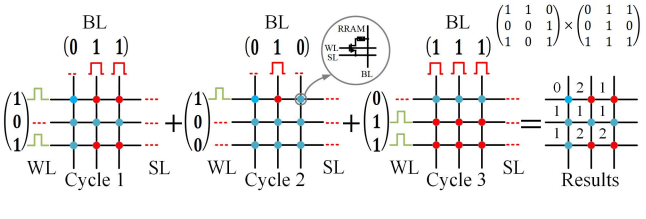

Fig.3 An example of calculating 3×3 matrix multiplication . During the ith cycle, parallel SET pulses corresponding to ith column of A and ith row of B are applied on the two sides of the cross-point array.

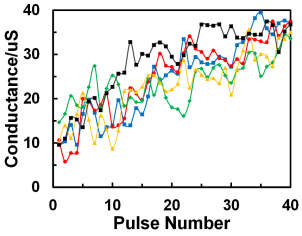

Fig.4 Device-to-device variation of RRAM analog behavior under consecutive SET pulses.

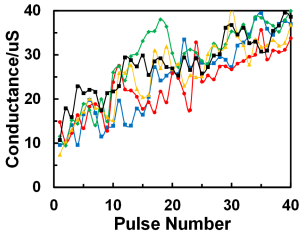

Fig.5 Cycle-to-cycle variation of RRAM devices' analog behavior under consecutive SET pulses

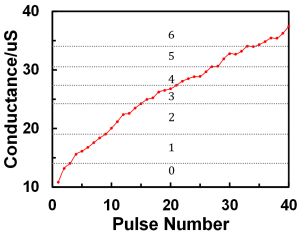

Fig.6 Average analog behavior for the 3×3 array.

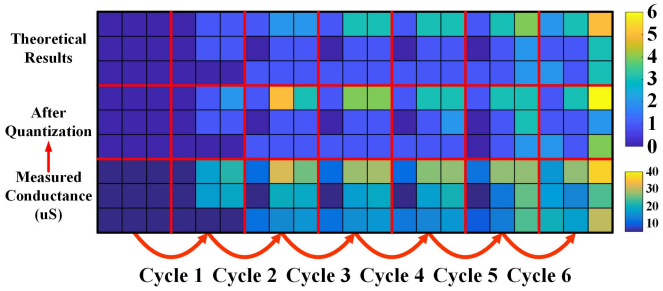

Fig.7 Measurement results for one example of matrices multiplication of $A \in \mathcal{B}^{3\times6}$ and $B \in \mathcal{B}^{6\times3}$, where $\mathcal{B}$ is Boolean space.

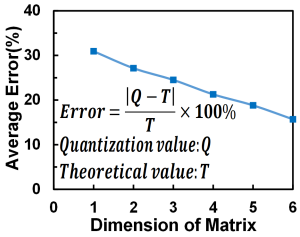

Fig.8 Measured average error against the dimension of matrix based on different input patterns.

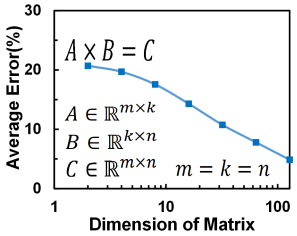

Fig.9 Simulated average error for different matrix dimensions based on stochastic RRAM model.

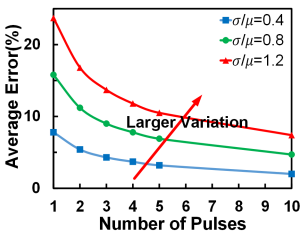

Fig.10 Average error against the number of pulses per quantized level for different variation.

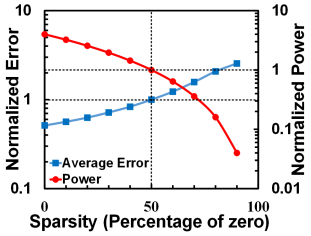

Fig.11 Normalized average error and power consumption against the sparsity of input matrices.

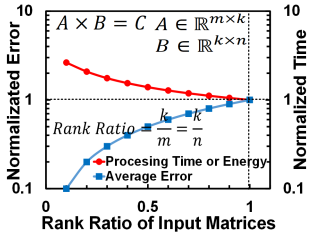

Fig.12 Normalized processing time (energy) and error against the rank ratio of input matrices.

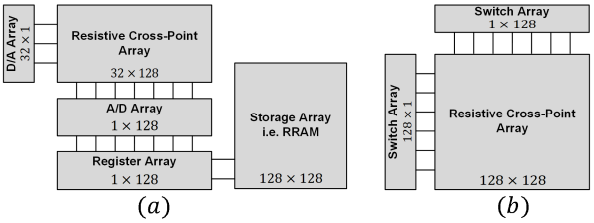

Fig.13 System configurations for calculating $A \times B = C$, $A \in \mathcal{B}^{128\times32}$, $B \in \mathcal{B}^{32\times128}$ using cross-point memory,. (a) conventional cross-point based Vector-Matrix Multiplication (VMM). (b) proposed in-memory Matrix-Matrix Multiplication (M2M).

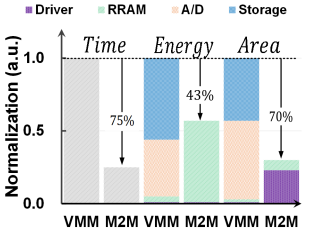

Fig.14 Processing time, energy and area comparison between VMM and M2M. The average resistance of RRAM is 100 kΩ.

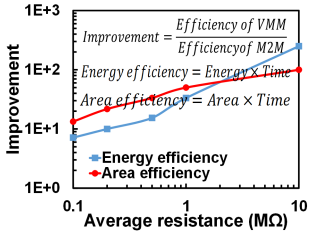

Fig.15 Energy and area efficiency benefits of M2M against the average resistance of RRAM.