HDC-IM: Hyperdimensional Computing In-Memory Architecture based on RRAM

Jialong Liu, Mingyuan Ma, Zhenhua Zhu, Yu Wang, Huazhong Yang
Department of Electronic Engineering, BNRist, Tsinghua University, Beijing, China

***Abstract*—Brain-inspired Hyperdimensional Computing (HDC) is a fast and robust classification algorithm, which works by mapping low-dimensional features to high-dimensional vectors and comparing distance in a high dimensional space. However, in traditional Von Neumann architecture, HDC causes high energy consumption because of large data movements between processor and memory. In this paper, we propose HDC-IM, a Hyperdimensional Computing-In-Memory architecture based on Resistive Random-Access Memory (RRAM), to boost the energy efficiency of HDC. HDC-IM puts computations in or near memory, which eliminates most of the data movements, providing a solution to reduce the energy consumption. In addition, to improve the computing parallelism, we use in-crossbar RRAM-based logic design to process encoding operation in HDC. The experimental results show that HDC-IM provides more than 100× speedup and higher energy efficiency compared with HDC on CPU. Moreover, in comparison with existing RRAM-based Neural Network accelerators, HDC-IM is more fault-tolerant taking into account RRAM device faults, achieving 20% higher accuracy than RRAM-based DNN on ISOLET dataset when 20% RRAM devices suffer from Stuck-At-Faults (SAFs).**

I. Introduction

In recent years, machine learning algorithms have played an increasingly important role in life. Convolutional Neural Network (CNN) has a good performance in various fields such as image recognition, target detection, and semantic analysis. However, the current trend of edge computing requires low delay and energy cost, while CNN has a great consumption of time and energy.

There are two key factors in achieving low-energy calculations at the edge. One is to propose algorithms with lower energy consumption and faster speed, and the other is to design devices with high energy efficiency and small area. For the former problem, the Hyperdimensional Computing (HDC) [1] proposed in recent years is a possible solution, for the latter problem, the computational framework based on Resistive Random Access Memory (RRAM) is a good choice.

In this work we propose an RRAM-based Hyperdimensional Computing-In-Memory Architecture: HDC-IM, which puts computations in memory to eliminate most of the data movements. We use in-crossbar RRAM computation to achieve high parallelism in data processing. In addition, we can effectively tackle the challenge of processing very-long vectors by proposing block encoding and mapping method. Finally, we prove HDC-IM to be robust taking into account RRAM device faults.

II. Preliminary

A. Hyperdimensional Computing

The main idea of Hyperdimensional Computing is mapping the features into hyperdimensional vectors, called Hypervectors (HVs), and making inference by comparing distance in high-dimensional space. HDC is applied in many classification problems [5] [6] [7]. The HDC algorithm for classification problem is shown in Fig. 1. In the train phase, we encode the entire training dataset into hyperdimensional space and get HV that represent every specific class. In the inference phase, we first encode the input data into HVs by using the same encoding method in the training phase, then compare the Hamming distance of the encoding result and every HV that represents a class. In this way we can get the classification result only using simple operations: bitwise XOR, addition and Hamming distance comparison.

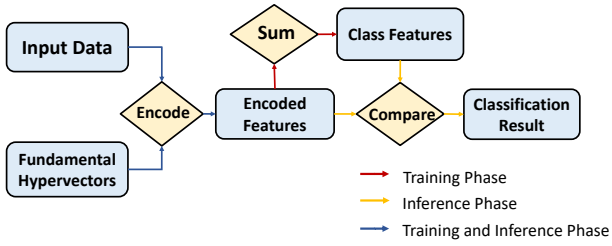

Fig. 1. HDC algorithm for classification problem

The main problem in HDC is the huge energy consumption caused by data movements between processors and memory, which makes it challenging to perform HDC at the edge. Thus, there are urgent needs for developing new computer architectures to improve the energy efficiency of Hyperdimensional Computing.

B. RRAM: computation, logic and problems

RRAM is an emerging device that stores data through variable resistance. RRAM devices are non-volatile and can support in-memory computing architecture design with high integration density, high access speed, and low power consumption. [2]

The RRAM crossbar can reduce the complexity of matrix vector multiplication from $O(n^2)$ to $O(1)$, which makes it easy to implement the bitwise addition of multiple vectors in HDC. Besides, in recent years, there have been a lot of work on the logical design of RRAM, such as MAGIC [3], IMPLY [4], etc. These methods can perform logical operations in parallel on the whole line, which can realize bitwise XOR in HDC. With these operations, it is possible to use RRAM for in-memory and high-parallelism HDC.

However, due to manufacturing process problems, RRAM has a high probability of device faults. Researchers have found that the probability of Stuck-At-Faults (SAFs) in RRAM

devices can reach 10%-20% [13]. In this case, the accuracy of the general NN algorithm will be greatly reduced, if the device's SAFs reach 20%, the error rate of CNN on the MNIST dataset will rise by more than 80% [13]. Compared with NN, the redundancy of HDC is higher, which is more conducive to overcome the accuracy loss caused by device faults.

III. Encoding Method Adapted to RRAM

When encoding the input data, we need Hypervectors that represent feature IDs and corresponding values. For each feature ID we use an HV to represent it, these HVs $(ID_1, ID_2, \ldots, ID_N)$ are randomly generated to maintain orthogonality and stored in Item Memory (IM). For input values, we first quantize the value into several intervals. To generate M D-dimensional HVs $(L_1, L_2, \ldots, L_M)$ representing M intervals, we have to ensure that the HVs corresponding to the adjacent quantization intervals are similar. The original generation method is that we generate the first D-dimensional random vector (L_1) representing the first interval, and for L_{n+1} we randomly flip $\frac{D}{M-1}$ bits of L_n. We store these HVs in Continuous Item Memory (CIM). When we get IM and CIM, we can encode any input data into hyperdimensional space. When data is input, for each feature we select the HV that represents the corresponding value from CIM, and we denote these HVs as $(v_1, v_2, \ldots, v_N)$. Then we can get the encoding result Hv of the input data using (1),

$$Hv = \sum_{i=1}^{N} ID_i * v_i, \quad v_i \in \{L_1, L_2, \ldots, L_M\} \tag{1}$$

where the operator '$*$' here means bitwise XOR operation. Fig. 2 shows the encoding process in HDC.

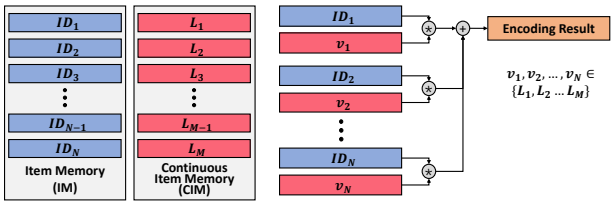

Fig. 2. The encode process of Hyperdimensional Computing

However, in the above encoding method, the operation between Hypervectors must be processed by blocks, but only one block can be processed at the same time. Whenever the group changes, new data needs to be re-selected from IM and CIM, which causes lots of data movements. Besides, in order to store a large number of random patterns, we need a dedicated memory, which makes us unable to implement the in-memory computing architecture.

Considering the characteristics of RRAM crossbar operations, we propose a block generation mode for IM and CIM. For a D-dimensional Hypervector ($D = 8192$ for example), suppose we can only process d dimensions ($d = 256$ for example) at a time. For $(ID_1, ID_2, \ldots, ID_N)$, we first generate N d-dimensional random vectors for N ID. After that, we only need to change the mapping method to make continuous shifting and splice the entire Hypervectors. Since the result obtained by shifting a random vector is substantially orthogonal to the original vector, $(ID_1, ID_2, \ldots, ID_N)$ obtained in this way can maintain approximate orthogonal characteristics to each other. For $(L_1, L_2, \ldots, L_M)$, we use the original flip method to generate M d-dimensional vectors, and then we continue to replicate and catenate the d-dimensional vectors to get the whole CIM. The original generating algorithm and our block generating algorithm are shown in Fig. 3.

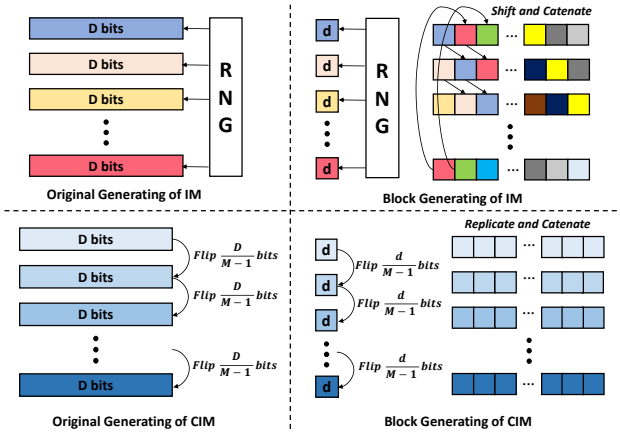

Fig. 3. The original and our new algorithm of generating IM and CIM

By using the proposed block generating algorithm, we only need to generate and store $(M + N)d$ bits, instead of $(M + N)D$ bits in original generating method for N features and M quantization intervals. In this way, the required random patterns are greatly reduced. At the same time, this grouping mode is well suited for RRAM in-crossbar processing. We only need to put these d-dimensional vectors into the processing units. By constantly changing the mapping method, we can complete the D-dimensional operations using the d-dimensional processing unit with almost no data movement. Table. I compares the group generation method with the traditional method on the ISOLET dataset [8]. It can be seen that the group generation method does not cause the classification accuracy loss.

TABLE I
THE PERFORMANCE OF ORIGINAL AND BLOCK ENCODING METHOD ON ISOLET DATASET

	Original Encoding Method	**Block Encoding Method**
Accuracy	86.0%	86.8%

IV. HDC-IM: HyperDimensional Computing In-Memory Architecture

A. HDC-IM Overview

The overall architecture is shown in Fig. 4(a), CIM and IM are distributed in multiple parallel processing units. When we get the input data, different input features are mapped to different operation units to complete the combination of the feature ID and the corresponding value. Then we sum these

vectors to complete encoding. The encoded result either enters the HV Memory (In the training phase) or the Classifier (In the inference phase). By changing the mapping between the input and each arithmetic unit, we can complete the entire HDC algorithm without data movements of IM and CIM.

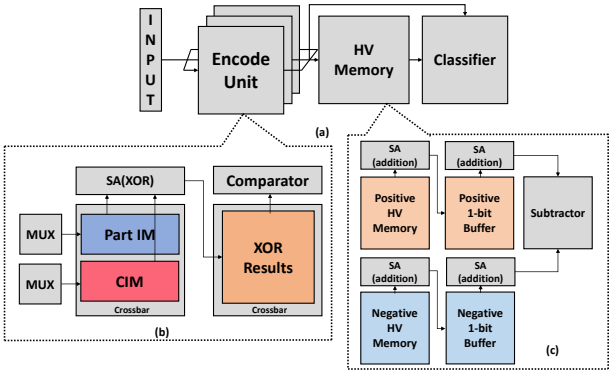

Fig. 4. (a): HDC-IM overview (b):Encoding Module (c):HV Memory Module

B. Encoding

In the encoding module we need to perform bitwise XOR and vector summation. For bitwise XOR operations, RRAM has a number of different ways to implement logic operations, and we choose Scouting Logic in our design [14]. In this method, we activate two wordlines concurrently and compare the current on the bitline with two thresholds. If the current is between the two thresholds, the result of XOR is 1, otherwise the result is 0. Compared with other methods such as IMPLY and MAGIC, the advantage of this method is that XOR operation can be realized in only one step, saving area and time, and reducing the number of RRAM writing operations. For the summation operation, we use the RRAM matrix-vector-multiplication design, which activates the corresponding wordlines that need to participate in the summation. Then we can directly get the result of the summation. In one single crossbar, only two rows of logic operations can be implemented at the same time. We can change the degree of parallelism by changing the number of encoding units used.

C. Classifying

We still use the RRAM matrix-vector-multiplication method to sum Hypervectors from the same class, but there exist two problems. The first one is that the dataset is too large to store in some cases, and the second problem is that we need to support the subtraction operation in retraining.

In order to solve the first problem we add the buffer module, which receives the results of the partial summation after they are obtained. The experiment proves that we only need to use 1 bit buffer here. Table. II shows the precision changes under different buffer bitumber on the ISOLET dataset (We shuffle the dataset and process 512 pieces of data at a time). From the results we know that the 1 bit buffer only causes a small loss of accuracy.

For the second problem, we need to add crossbars that stores negative numbers in the memory module and subtract the two parts of the current to get the final result. The structures of HV memory module is shown in Fig. 4. In the classifying module, we use multiple Winner-Takes-All (WTA) blocks in a tree structure which can easily compare the Hamming distance by choosing the maximum current.

TABLE II
THE PERFORMANCE OF HDC-IM WITH DIFFERENT BUFFER BITNUMBER (ON ISOLET DATASET)

Buffer Bitnumber	Accuracy
1	86.4%
2	86.4%
4	86.4%
8	86.7%
Full Precision	86.8%

V. EXPERIMENTAL RESULTS

A. Experiment Setup

We use behavioral level simulation to simulate the process in HDC-IM. The parameters we used refer to [9] [10] [11] [12] [14]. In order to analyze the performance of HDC-IM, we first consider the HDC running on the CPU as a baseline, we also compare the speed and power consumption of this architecture with a simple DNN after GPU acceleration. In addition, to test the robustness of HDC-IM, we compare this architecture with the DNN based on the RRAM neural network accelerator to analyze their performance differences in fault tolerance.

B. Performance Analysis

We first compare the performance of HDC-IM, CPU-based HDC, and GPU-based DNN. In terms of the parameter settings of HDC-IM, We set $M = 12$, $D = 8192$ and $d = 256$, using 32 encode units to process parallel HDC encoding. The results are shown in Table. III. The DNN model we use has 5 layers with (617, 512, 1024, 1024, 26) parameters for each layer and the dataset we used is ISOLET.

It can be seen that HDC-IM has a low time and energy consumption of 0.19s and 0.12J, respectively, without retraining, which is 500 times faster than HDC on the CPU and exceeds 10^4 improvement on energy efficiency, but the accuracy is lower than the DNN algorithm. When considering retraining, HDC has similar accuracy with the GPU-accelerated simple DNN, and has a speed increase of 2 times and a power consumption reduction of 104 times.

TABLE III
PERFORMANCE COMPARISON OF DIFFERENT ALGORITHMS AND ARCHITECTURES ON ISOLET DATASET

	Accuracy	Delay(s)	Energy(J)
HDC-IM(without Retrain)	86.4%	0.19	0.12
HDC-IM(with Retrain)	92.1%	3.86	2.28
HDC on CPU(without Retrain)	86.4%	93	2760
HDC on CPU(with Retrain)	92.1%	132	3960
DNN on GPU	94.5%	7.9	239

C. Robustness Analysis

We consider Stuck-At-Faults (SAFs) in RRAM devices, which mean that the RRAM cell stucks at HRS or LRS. Since each step in HDC-IM is done on the RRAM, errors can occur from read, write, XOR and addition operations. We add SAFs

with a certain probability in every step by behavioral level simulation. We use the 4-bit quantized DNN (the structure is the same as we used before) as baseline, which can still achieve an accuracy of 92.5% on ISOLET dataset without considering SAFs. We add randomly generated SAFs to the weights. We suppose the SAFs in RRAM crossbars follows an uniform spatial distribution.

Fig. 5. shows the accuracy change of two algorithms when the probability of SAFs changes from 0 to 20%. In each case we performed 100 experiments and obtained the mean, maximum and minimum values. The error bars in the graph indicate the fluctuation range of the results. It can be seen that the DNN is slightly more accurate than HDC when the device has no obvious error, but when the probability of the SAFs reaches 0.1 or more, the performance of the DNN algorithm will be significantly reduced. In the case of a SAFs probability of 0.2, DNN can only achieve an average correct rate of 35%, while HDC can still achieve an average correct rate of 55%. When considering the variance of performance in each case,

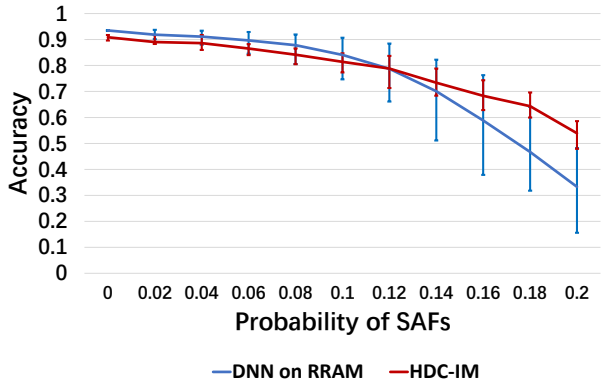

Fig. 5. Performance of HDC-IM and RRAM-based DNN with different SAFs probability

HDC is obviously a relatively stable algorithm, while the DNN algorithm is very unstable in the case of large SAFs. That's because only a few weights have a big impact on the results. That is to say, the importance of the weights of different positions of the DNN is different. However, for HDC, since each position is equally important, its accuracy has little to do with the error distribution, and always maintains a normal performance.

VI. Conclusion

In this paper we present a RRAM-based Hyperdimensional Computing architecture, HDC-IM. Considering the processing-in-memory characteristics of RRAM, we propose block encoding method, so that the very-long vector operations in HDC can be efficiently processed in groups without huge overhead caused by data movements. In terms of architecture design, we propose an encoding implementation based on Scouting logic method and RRAM matrix-vector-multiplication to improve the parallelism of the encoding process. We also add a 1-bit buffer module and crossbars representing negative numbers to solve the problem of oversized datasets and support the operation of retraining. Experiments show that this architecture has superior speed and energy efficiency in processing HDC, and can even be compared with simple neural networks on GPUs. HDC-IM's fault-tolerant performance comes from the redundancy of the Hyperdimensional Computing itself, which makes it possible to compensate for the problems caused by RRAM device errors in some degree. Experiments also show that HDC-IM has a significant advantage over RRAM-based DNN in the performance considering high probability of RRAM device faults. We expect Hyperdimensional Computing to have a wider range of uses in the future.

VII. Acknowledgements

This work was supported by National Key Research and Development Program of China (No. 2017YFA0207600), National Natural Science Foundation of China (No. 61720106013, 61832007, 61622403, 61621091), Beijing National Research Center for Information Science and Technology (BNRist), and Beijing Innovation Center for Future Chips.

References

[1] Rahimi, P. Kanerva, L. Benini and J. M. Rabaey, "Efficient Bio signal Processing Using Hyperdimensional Computing: Network Templates for Combined Learning and Classification of ExG Signals," in Proceedings of the IEEE, vol. 107, no. 1, pp. 123-143, Jan. 2019.

[2] Gu P, Li B, Tang T, et al., "Technological exploration of rram crossbar array for matrix-vector multiplication". The 20th Asia and South Pacic Design Automation Conference (ASPDAC), 2015. 106-111.

[3] S. Kvatinsky et al., "MAGIC-Memristor-Aided Logic" in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 61, no. 11, pp. 895-899, Nov. 2014.

[4] 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" in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 10, pp. 2054-2066, Oct. 2014.

[5] Rahimi et al., "A robust and energy efcient classifier using brain-inspired hyperdimensional computing," in ISLPED, August 2016.

[6] M. Imani, D. Kong, A. Rahimi and T. Rosing, "VoiceHD: Hyperdimensional Computing for Efficient Speech Recognition," 2017 IEEE International Conference on Rebooting Computing (ICRC), Washington, DC, 2017, pp. 1-8.

[7] M. Imani, Y. Kim, T. Worley, S. Gupta and T. Rosing, "HDCluster: An Accurate Clustering Using Brain-Inspired High-Dimensional Computing," 2019 Design, Automation Test in Europe Conference Exhibition (DATE), Florence, Italy, 2019, pp. 1591-1594.

[8] "Uci machine learning repository." http://archive.ics.uci.edu/ml/datasets /ISOLET.

[9] R. Dlugosz, A. Rydlewski and T. Talaska, "Low power nonlinear Min/Max filters implemented in the CMOS technology" 2014 29th International Conference on Microelectronics Proceedings - MIEL 2014, Belgrade, 2014, pp. 397-400.

[10] Khorami, M. B. Dastjerdi and A. F. Ahmadi, "A low-power high-speed comparator for analog to digital converters," 2016 IEEE International Symposium on Circuits and Systems (ISCAS), Montreal, QC, 2016, pp. 2010-2013.

[11] Shimeng Yu, "Resistive Random Access Memory (RRAM)," in Resistive Random Access Memory (RRAM) , Morgan Claypool, 2016, pp.

[12] H. -. P. Wong et al., "Metal-Oxide RRAM," in Proceedings of the IEEE, vol. 100, no. 6, pp. 1951-1970, June 2012.

[13] L. Xia et al., "Stuck-at Fault Tolerance in RRAM Computing Systems," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 8, no. 1, pp. 102-115, March 2018.

[14] L. Xie et al.,"Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing," 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Bochum, 2017, pp. 176-181.