EasyChair Preprint

№ 13938

A Novel Efficient Maximum Searching Algorithm in ReRAM Array

Wenqing Wang, Ziming Chen, Quan Deng and Liang Fang

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

July 11, 2024

A Novel Efficient Maximum Searching Algorithm in ReRAM Array

Wenqing Wang* , Ziming Chen*, Quan Deng✉, and Liang Fang

National University of Defense Technology, Changsha, 410073, China
{wangwenqing, chenziming18, dengquan12, lfang}@nudt.edu.cn

Abstract. Computing in Memory (CIM) is a promising architecture that accelerates applications by eliminating the data movement between memory and processing units. Memristor-Aided Logic (MAGIC) offers massive parallelism, flexible computing, and non-volatility. We propose a novel algorithm to find the maximum value in a set of numbers, leveraging the MAGIC NOR gate and the parallel structure of memristor arrays. Our experiment result shows that the proposed algorithm is 3.2× faster than AritPIM for finding the maximum of 8-bit value in 512 numbers. Additionally, this algorithm can accelerate the accumulation of floating-point vector multiplication.

Keywords: Maximum · MAGIC NOR gate · Memristor array.

1 Introduction

Emerging applications, such as Machine Learning, Large Language Models (LLM) , and Graphic Processing are developing rapidly, requiring both significant computational and memory resources. Computing in Memory (CIM) is a promising architecture to accelerate these applications by eliminating the data movement between memory and processing units. Memristor-Aided Logic (MAGIC) NOR [1] achieves massive parallelism, flexible computing, and non-volatility. Finding the maximum in a set of numbers is a common computational problem. For example, we need to find the maximum exponent in a set of floating-point numbers in the floating-point accumulation process. We propose a novel algorithm that utilizes the MAGIC NOR gate and the parallel structure of memristor arrays to find the maximum in a set of numbers. Our experiment result shows that the proposed algorithm is 3.2× faster than the traditional tree-based parallel comparison method for finding the maximum of 8-bit value in 512 numbers.

* Wenqing Wang and Ziming Chen contribute equally to this work.

✉ Quan Deng is the corresponding author.

Supported by NSFC-62172155, U22A2027,NSFC-62202481, ZK22-05 and 22-TDRCJH-02-006.

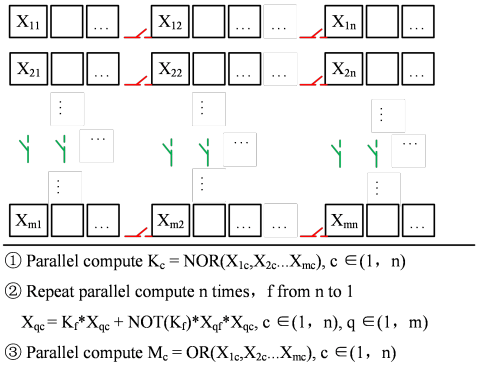

Fig. 1. The proposed algorithm to find the maximum number computation in the 512×512 ReRAM array. Red represents row transistors and green represents column transistors.

2 Maximum Searching Algorithm

The conventional way to find the maximum in a set of numbers is to compare numbers one by one. We propose a novel algorithm to find the maximum number that leverages the ReRAM array's logic function and parallelism. The key idea is to compare numbers from the most significant bit (MSB) to the least significant bit (LSB). If there is one and only one number whose MSB is 1, this number is the maximum. Then, set others to 0. If there is more than one number with MSB being 1, set the other numbers with 0 in MSB to 0. Then, compare the next bit among the remaining numbers. If MSBs of all numbers are 0, compare the next bit. The next bit is compared in the same way as MSB. After comparing from MSB to the LSB, only the maximum number remains.

The proposed algorithm to find the maximum number computing process in a 512×512 ReRAM array is illustrated in Fig.1. Each array row is divided into 32 partitions by row transistors, and each column is divided into 64 partitions by column transistors. The numbers are stored row by row in the ReRAM array with each bit occupying a separate partition within each row. Firstly, compute the NOR of all the same position bit of these numbers, obtain the value K_f, where f represents the bit position; Secondly, for each bit X_{qc} (representing number X_q in position c bit) of these numbers, the operation $\mathrm{K}_f\mathrm{X}_{qc} + \mathrm{NOT}(\mathrm{K}_f)\mathrm{X}_{qf}\mathrm{X}_{qc}$ is performed. K_f is 1 indicating all numbers in the f position are 0. The value of each bit remains unchanged. K_f is 0, indicating that there is at least one number in the f position that is 1. If X_{qf} is 1 (indicating number X_q in position f bit is 1), the value of this number remains unchanged. If X_{qf} is 0 (indicating number X_q in position f bit is 0), then the number X_q will be set to 0. Following the computation from the highest position to the lowest position, all numbers will be 0 except the largest one remains. Finally, perform an OR operation on all bits at the same position across these numbers, and the resulting value will indicate the largest number. The original X_{qc} is still retained without overwriting. Due

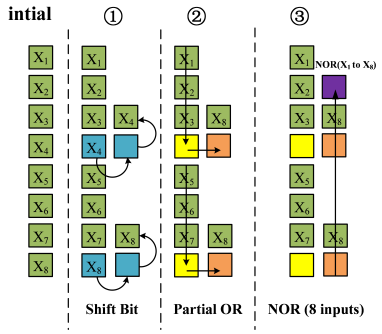

Fig. 2. Multiple inputs of NOR operation within a column.

to the NOR being complete logic, the above logic function can be combined with NOR logic.

The computing process of finding the maximum number in the array involves a multitude of inputs NOR/OR operations in columns, as well as broadcast, and parallel computing of logic functions in partitions. The broadcast and parallel computing method in partitions is detailed in MultPIM[3]. Regarding multiple inputs NOR operations, ideally, the inputs could number in the hundreds, as described in [1]. However, considering practical circuit constraints, we implement NOR operations with up to 4 inputs to parallelize the computation of a column's NOR value. The process of parallel computing multiple inputs NOR at columns is shown in Fig.2. The initial state is each bit within each partition is in the same columns. Firstly, a bit is shifted to another column by performing a double NOT operation to create space for the NOR operation within the column. Secondly, the result of the partial NOR operation is moved to another column by a NOT operation, and the result is a partial OR. Thirdly, the value of the partial OR is NOR-ed with the shifted bits to obtain the result of an 8-input NOR. Since the array is column-partitioned, we can parallelize these operations. Fourthly, using the partial result succeeds parallel NOR operation, we obtain the NOR of position q bit. All positions NOR of the entire column value can be computed in parallel. The operation of multiple inputs of OR is the NOT of multiple inputs of NOR operation. In addition, since this method destroys the column data, if the column data is needed, it must first be copied to another column.

To demonstrate the efficiency of the proposed algorithm for determining the maximum of group numbers, we compare it with the state-of-the-art AritPIM bit-parallel subtraction approach [2], which is inspired by parallel-prefix carry-look ahead adders with a tree-based parallelism structure. We identify the maximum of 8-bit values in 512 numbers using the proposed algorithm, which is 3.2 $\times$ faster than the state-of-the-art AritPIM bit-parallel subtraction approach [2] with a tree-based parallelism structure. This speedup is due to the algorithm taking full advantage of array parallelism and logical capabilities. Fig.3 shows the latency of floating-point vector multiplication by the naive design (AritPIM

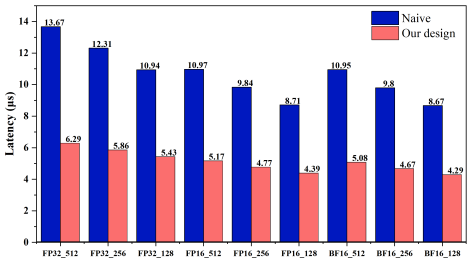

Fig. 3. The latency of Floating-Point Vector Multiplication by naive design (AritPIM bit-parallel subtraction approach with a tree-based parallelism structure) and our design.

bit-parallel subtraction approach with a tree-based parallelism structure) and our design with the switching latency of the memristor set to $1ns$ [4].

3 Conclusion

We present a novel algorithm optimized for memristor array operations to find the maximum number in a set of numbers efficiently. This algorithm leverages inherent parallelism and logical capabilities of the memristor array to achieve a significant speedup over traditional methods. The experiment result shows that the proposed algorithm is 3.2× faster than the state-of-the-art AritPIM bit-parallel subtraction approach when identifying the maximum 8-bit value in 512 numbers. This work finding the maximum in a set of numbers can accelerate the floating-point accumulation process, as it requires determining the maximum exponent in the set of floating-point numbers.

References

1. Kvatinsky, S., Belousov, D., Liman, S., Satat, G., Wald, N., Friedman, E.G., Kolodny, A., Weiser, U.C.: Magic—memristor-aided logic. IEEE Transactions on Circuits and Systems II: Express Briefs **61**(11), 895–899 (2014)
2. Leitersdorf, O., Leitersdorf, D., Gal, J., Dahan, M., Ronen, R., Kvatinsky, S.: Aritpim: High-throughput in-memory arithmetic. IEEE Transactions on Emerging Topics in Computing **11** (2023)
3. Leitersdorf, O., Ronen, R., Kvatinsky, S.: Multpim: Fast stateful multiplication for processing-in-memory. IEEE Transactions on Circuits and Systems II: Express Briefs **69**(3), 1647–1651 (2022). https://doi.org/10.1109/TCSII.2021.3118215
4. Truong, M.S.Q., Chen, E., Su, D., Shen, L., Glass, A., Carley, L.R., Bain, J.A., Ghose, S.: Racer: Bit-pipelined processing using resistive memory. MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (2021), https://api.semanticscholar.org/CorpusID:239011568