Accelerating Multiplication and Parallelizing Operations in Non-Volatile Memory

Mohsen Imani, Saransh Gupta, and Tajana Rosing
CSE, UC San Diego, La Jolla, CA 92093, US

This is based on paper [1], presented at Design Automation Conference (DAC) 2017. You can access to the full paper at: https://doi.org/10.1145/3061639.3062337

I. Processing in-Memory

Today's IoT applications often analyze raw data by running machine learning algorithms such as classification or neural networks in data centers [2], [3]. Sending the entire data to cloud for processing is not scalable and cannot guarantee the required real-time response [4]. Running data intensive workloads with large datasets on traditional cores results in high energy consumption and slow processing speed, majorly due to the large amount of data movement between memory and processing units [5], [6]. The idea of processing in-memory aims to reduce this problem by avoiding the memory/cache bottleneck. It reduces data movement by processing data within the memory, thus improving both performance and energy efficiency [7], [6].

On the other hand, most of the algorithms running on today's sensors' data are statistical in nature, and thus do not require exact answers [8], [9]. Similarly, in audio and video processing we have long exploited the fact that humans do not perceive all the colors or sounds equally well. Approximate computing is an effective way of improving the energy and performance by trading some accuracy. However, most of the existing techniques provide less energy or performance efficiency due to considerable data movement and lack of configurable accuracy.

In this paper, we propose a configurable approximate processing in-memory architecture, called APIM, which supports addition and multiplication operations inside the non-volatile RRAM-based memory. APIM exploits the analog characteristic of the memristor devices to enable basic bitwise computation and then, extend it to fast and configurable addition and multiplication within memory. This would make processing machine learning and data analysis algorithms more efficient since most of them use addition and multiplication aggressively. We propose a blocked crossbar memory which, along with the configurable interconnects, introduces flexibility in executing operations and facilitates shift operations in memory. Then, we introduce a novel approach for fast addition in memory. Finally, we design an in-memory multiplier using the proposed memory unit and fast adder. For each application, APIM can dynamically tune the level of approximation in order to trade the accuracy of computation while improving energy and performance. Our experimental evaluation over six general OpenCL applications shows that the proposed design improves the performance by 20× and provides 480× improvement in energy-delay product while ensuring less than 10% average relative error.

A. APIM Architecture

A single-block memory crossbar architecture allows easy copying of data provided the source and destination are in the same column/row. While being acceptable in many cases, this memory structure limits the performance of instructions which involve a lot of shifting and asymmetric movement of data. One such instruction is multiplication where the multiplicand is shifted and added. Multiple copy operations can emulate a shift operation. However, such an approach is impractical when the number to be shifted is large since it requires shifting each and every bit individually. The problem is aggravated when multiple such numbers are to be shifted.

We hence propose the use of a blocked memory structure as shown in Figure 1(a). The crossbar is divided into blocks. Any new data which is loaded into the memory is stored in the data block. Whenever there is a request to process data, it is copied to the processing block and computation is done using MAGIC [10], [11]. The two blocks are structurally the same and can be used interchangeably. These blocks are connected by configurable interconnects. The interconnects support shift operations inherently such that i^{th} bitline of one block can be connected to $(i+j)^{th}$ bitline of another block. The availability of interconnects allows the memory to shift data while copying it from one block to another without introducing any latency overhead. This makes shifting an efficient operation since the entire string of data can be shifted at once, unlike shifting each bit individually.

B. Multiplication in APIM

The process of multiplication is divided into three stages, partial product generation, fast addition, and final product generation as shown in Figure 1. The partial product generation stage creates partial products of a $N \times N$ multiplication. These partial products are then added using a fast in-memory carry save adder (CSA). The fast addition reduces N numbers to 2. The final product generation stage adds two numbers generated by the previous stage and outputs the product of $N \times N$ multiplication.

We propose the use of sense amplifiers to develop a faster partial product generator as shown in Figure 1(b). The design exploits the fact that the partial product is the multiplicand itself if multiplier bit is '1' and 0 otherwise. M_2 is read bit-wise using the sense amplifier. If the read bit is '1', M_1 is copied, while nothing is done when the bit is '0'. We now use an in-memory carry save addition to reduce the number of partial products to 2. Since a step in the fast adder involves parallel additions, it requires that the three addends of a 3 : 2 adder are present in the same columns (rows) and all such groups in a step are present in the same rows (columns). This arrangement can be easily achieved by the interconnects while copying data. The major advantage of reduction addition is that the time taken by this adder is independent of the size of the operands *i.e.*, $N \times 32$ multiplication takes the same time in this stage for any value of N. The final product generation stage adds the two outputs of the previous stage to generate the required product.

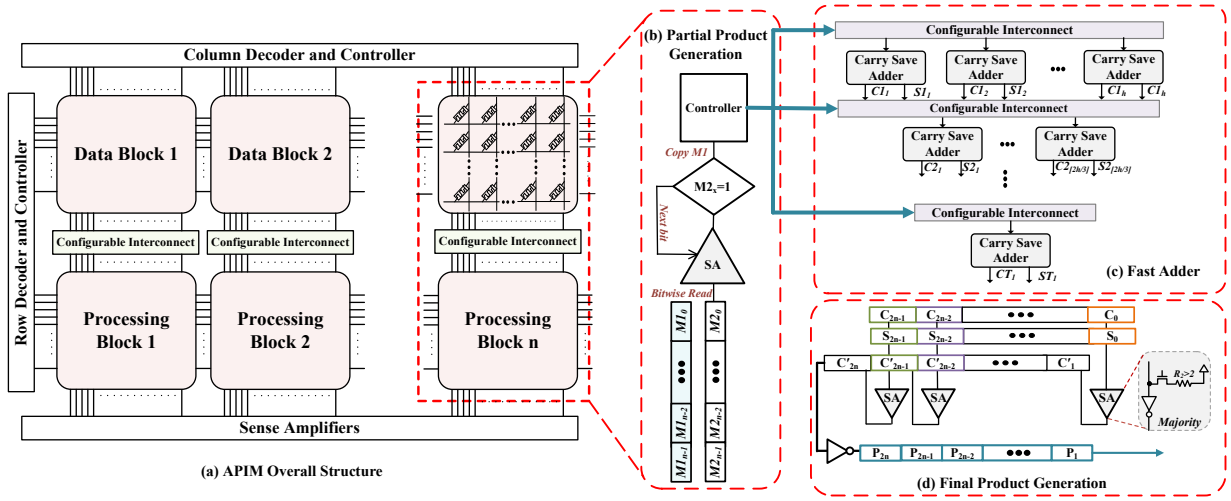

Fig. 1. (a) The overall structure of APIM consisting of several data and processing blocks. (b) the APIM controller and parallel product generator. (c) Fast adder three structure consisting of Carry save adder and configurable interconnects. (d) Final product generator to ripple the carry bits of tree structure.

C. APIM Approximation

The individual additions in the final stage cannot occur in parallel since they require the propagation of carry in order to generate the final answer. The latency of this operation is dominant as compared to the previous stages of multiplication, making the last stage a bottleneck in the entire process. However, we can dramatically speed it up if a fully accurate result is not desired. This is the case with many highly data intensive applications which tolerate some inaccuracy as long as it is within the prescribed limits. Our design introduces approximation by exploiting the fact that the sum bit (S) of an 1-bit addition is the complement of the generated carry bit (C_{out}) except for two combinations of inputs (*i.e.*,(A,B,C) = (0, 0, 0) and (1, 1, 1)). It evaluates C_{out} accurately (hence, preventing the propagation of error) and then approximates S. Our design uses a modified sense amplifier (SA) which supports `MAJ` (majority) function.

II. Detailed Evaluations

We implement the APIM functionality by changing the multi2sim [12], cycle-accurate CPU-GPU processor. Performance and energy consumption of proposed hardware are obtained from circuit level simulations for a 45nm CMOS process technology using Cadence Virtuoso. Figure 2 compares the performance efficiency of the proposed design with the state-of-the-art prior work [11], [13]. The work in [11] computes addition in-memory using MAGIC logic family, while the work in [13] uses the complementary resistive switching to perform addition inside the crossbar memory. Our evaluation comparing the energy and performance of addition of N operands of length N bits each shows that the APIM can achieve at least 2× speed up compared to previous designs in exact mode. APIM can be at least 6× faster with 99.9% accuracy. We also compare our design with AMD Radeon R9 390 GPU. Our evaluation shows that for most applications using datasets larger than 200MB (which is true for many IoT applications), APIM is much faster and more energy efficient than GPU. With 1GB dataset, the APIM design can achieve 28× energy savings, 4.8× performance improvement as compared to GPU architecture.

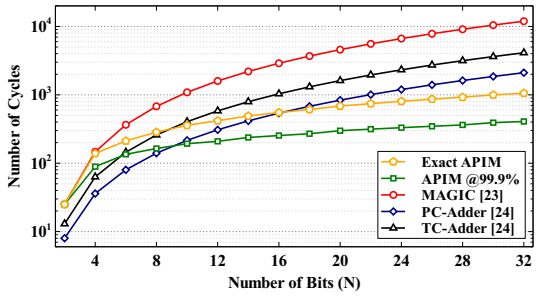

Fig. 2. Performance comparison of the proposed design with previous work for addition of N operands, each sized N bits.

References

[1] M. Imani *et al.*, "Ultra-efficient processing in-memory for data intensive applications," in *Proceedings of the 54th Annual Design Automation Conference 2017*, p. 6, ACM, 2017.

[2] J. Gubbi *et al.*, "Internet of things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[3] M. Samragh *et al.*, "Looknn: Neural network with no multiplication," in *IEEE/ACM DATE*, 2017.

[4] K. Hwang *et al.*, *Distributed and cloud computing: from parallel processing to the internet of things*. Morgan Kaufmann, 2013.

[5] R. Balasubramonian *et al.*, "Near-data processing: Insights from a micro-46 workshop," *Microarchitecture*, vol. 34, no. 4, pp. 36–42, 2014.

[6] G. Loh *et al.*, "A processing-in-memory taxonomy and a case for studying fixed-function pim," in *WoNDP*, 2013.

[7] A. M. Aly *et al.*, "M3: Stream processing on main-memory mapreduce," in *ICDE*, pp. 1253–1256, IEEE, 2012.

[8] J. Han *et al.*, "Approximate computing: An emerging paradigm for energy-efficient design," in *ETS*, pp. 1–6, IEEE, 2013.

[9] M. Imani *et al.*, "Efficient neural network acceleration on gpgpu using content addressable memory," in *IEEE/ACM DATE*, 2017.

[10] S. Kvatinsky *et al.*, "MAGIC – memristor-aided logic," *TCAS II*, vol. 61, no. 11, pp. 895–899, 2014.

[11] N. Talati *et al.*, "Logic design within memristive memories using memristor-aided loGIC (MAGIC)," *IEEE TNano*, vol. 15, pp. 635–650, jul 2016.

[12] R. Ubal *et al.*, "Multi2sim: a simulation framework for cpu-gpu computing," in *PACT*, pp. 335–344, ACM, 2012.

[13] A. Siemon *et al.*, "A complementary resistive switch-based crossbar array adder," *JETCAS*, vol. 5, no. 1, pp. 64–74, 2015.