Unlocking High Resolution Arithmetic Operations within Memristive Crossbars for Error Tolerant Applications

Kamalika Datta*, Saman Froehlich¶ Saeideh Shirinzadeh*†, Dev Narayan Yadav§ Indranil Sengupta‡§, Rolf Drechsler*¶
*German Research Centre for Artificial Intelligence (DFKI), Bremen, Germany
† Fraunhofer Institute for Systems and Innovation Research (ISI), Karlsruhe, Germany
‡Department of Computer Science and Engineering, JIS University, India
§Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India
¶Group of Computer Architecture, University of Bremen, Germany
E-mail: kamalika.datta@dfki.de, froehlich@uni-bremen.de, saeideh.shirinzadeh@dfki.de,
dev.narayan@iitkgp.ac.in, indranil.sengupta@jisuniversity.ac.in, drechsler@uni-bremen.de

***Abstract*—Memristor-based crossbar architectures have been explored by researchers for neuromorphic computing, where analog vector-matrix multiplication can be carried out in a single time step. In this paper we explore such architectures for carrying out various arithmetic operations. Since the computations are carried out in analog domain, they are affected by fabrication and performance variability of the manufactured devices. As a result, there can be inherent errors during the computation. However, the architecture can be suitable for approximate computing applications where some errors can be tolerated. We have proposed a method for carrying out arithmetic operations with any multiple of k-bit resolution on the crossbar, for some limited values of k. The fault tolerant capability of the proposed architecture is evaluated through experimentation on benchmark datasets. We also perform case studies to analyze the performance of the approach with particular emphasis on approximate computing. The results of the case studies show that certain applications indeed exhibit fault tolerance in presence of faulty memristors.**

***Index Terms*—Neuromorphic computing, approximate computing, vector processor, memristor crossbar**

I. Introduction

The history of computing has seen a transition from analog to digital in the quest for reliability, speed and accuracy. Advancements in digital computing have seen the emergence of faster and more capable processors to address the requirements of various compute intensive applications. In recent times it has been observed that there are applications where high levels of computational accuracy is not essential, and it is possible to achieve drastic reductions in the hardware required for implementation. This is often referred to as *Approximate Computing* (AC), and has drawn the attention of several researchers. This can be achieved at the level of algorithms, architecture, and also at the lowest levels of circuit primitives.

There have also been a lot of research on memristor-based resistive memory systems, which have manifested in innovative architectures like in-memory computing and neuromorphic computing. In particular, architectures based on memristive crossbars can drastically speed up vector-matrix multiplication operation that is an essential component in neuromorphic computing. In this paper, we have explored memristive crossbars for carrying out various scalar and vector arithmetic operations. Such computations are essentially carried out in analog domain where there can be some errors during computation due to device variability.

Such a computational engine can be very suitable for applications that are amenable to AC. The main contributions of the present work can be summarized as follows:

a) The detailed steps to perform scalar and vector addition, subtraction and multiplication on a crossbar are presented.
b) A technique to enhance the computational accuracy of the arithmetic operations is proposed, which uses additional crossbar resources.
c) The fault tolerance capability of the approach is evaluated on standard benchmarks.
d) The performance of the approach with regards to AC is evaluated with some case studies.

The rest of the paper is organized as follows. Section II presents a brief review of AC and use of memristor crossbar in neuromorphic computing. Section III surveys the capabilities of a crossbar for carrying out various arithmetic operations. Section IV presents an approach to enhance the computational precision. Section V evaluates the fault tolerance capability of the approach, while section VI presents the experimental evaluation on some benchmark applications. Finally, Section VII concludes the paper.

II. Literature Survey

We now briefly discuss about memristors and crossbar structures for neuromorphic computing, followed by a brief survey of relevant approximate computing (AC) approaches.

A. Memristors and Memristive Crossbar

A memristor is a two-terminal passive element that exhibits non-volatile resistive change in response to an applied volt-

age [1]. This non-volatile resistive switching property makes it suitable for realizing non-volatile resistive memories [2].

Memristors are typically fabricated in a crossbar structure, where each device is created at the junction of every pair of horizontal and vertical nanowires [3]. The basic crossbar is also referred to as a 0T1R (*zero transistor, one resistor*) structure. Memristor crossbars have been proposed for carrying out logic operations, for use as resistive storage, and also for neuromorphic computing [4], [5], [6], [7]. For logic and storage applications, the memristors are typically programmed in one of two states, *low resistance* and *high resistance*. In contrast, for neuromorphic applications, the resistive states of the memristors need to be multi-valued. A memristor is said to have k-bit resolution if it can be in one of 2^k resistive states.

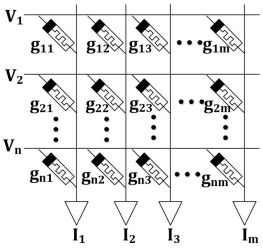


Fig. 1. Memristor crossbar for vector-matrix multiplication

One of the basic operations required in deep learning applications is the *vector-matrix multiplication (VMM)*. The schematic diagram of an $n \times m$ crossbar for carrying out VMM is shown in Fig. 1. Based on the applied voltages $V = \{V_1, V_2, \ldots, V_n\}$, the circuit generates the currents $I = \{I_1, I_2, \ldots, I_m\}$ as:

$$I_j = \sum_{i=1}^{n} (V_i \times g_{ij}), \quad \text{for } j = 1 \text{ to } m. \tag{1}$$

where g_{ij} represents the conductance of the memristor in row i and column j. It may be noted that $g_{ij} = \frac{1}{r_{ij}}$, where r_{ij} denotes the corresponding resistance [8]. The entire VMM operation is carried out in a single step.

Use of memristor crossbar for performing analog dot-product and VMM operations have been explored earlier (e.g., [9]). In this paper we analyze the fault-tolerant capabilities of the crossbar for multi-precision arithmetic with emphasis on approximate computing.

B. Neuromorphic Computing on Crossbar

Prezioso et al. [4] proposed the design of a neural network using memristive crossbar, and used a single-layer perceptron to perform classification of 3×3 black-and-white images. Xia et al. [8] analyzed the VMM operation considering various fabrication challenges and interconnect resistances. They analyzed the results using MNIST dataset, and also performed a resistance resolution test showing that 6-bit resolution can achieve 90% recognition accuracy. Hu et al. [10] used a 128×64 crossbar for performing VMM using high precision analog tuning using 6-bit memristors. They used memristors with 6-bit resolution and showed a huge improvement in computation efficiency with respect to the digital counterpart.

Although researchers have used memristive crossbars for neuromorphic applications, they can be used for carrying out arithmetic operations as well. In particular, they are suited for applications where AC can be used.

C. Approximate Computing (AC) Approaches

AC is a design paradigm that tries to trade off accuracy for performance. Research studies have shown that many applications are inherently error tolerant, e.g. image processing and machine learning. As memristor fabrication is still very error prone, such error-tolerant applications are excellent candidates for use-cases where memristors can be used. In the following, we review some of the AC related works for such applications.

Many prior works have used image processing to show the effects of their AC designs. In [11], an *accuracy-configurable adder* (ADA) has been proposed for approximate arithmetic designs, by cutting the carry chain of the adder. The authors have used image smoothing with Gaussian filters to show the applicability of their adder. The proposed design is compared with other approximate adder designs and the *Peak-Signal-To-Noise Ratio* (PSNR) is used to assess the quality of the resulting image. The authors of [12] discuss a processor that uses a look-up table to approximate floating-point operations, and edge detection algorithms are used to evaluate the results. The authors of [13] present a k-Nearest Neighbor (kNN) approach to approximate floating point multipliers, which are then applied to image processing by using Gaussian filters. Finally, in [14] the authors propose a rewriting scheme for *AND-Inverter Graph* (AIG) based synthesis, where a cut-based approach is used to identify parts of the AIG that can be approximated. They have used image compression to demonstrate the approach in a real-word application.

AC has also been successfully applied to machine learning algorithms. In [15] approximate multipliers are used in neural network applications, where an evolutionary approach is used to identify the multiplier to be used in each layer of the network. The weights of the neural networks are quantized to 8-bit fixed-point numbers. The authors of [16] used an approximate multiplier for implementing a kNN classifier. It is shown that using the approximate multiplier the accuracy drops only slightly, but the energy-delay-product is improved.

III. Arithmetic Operations on the Crossbar

The power of memristor crossbar as VMM computation engine can be leveraged for carrying out various (approximate) arithmetic computations. These are discussed in the following subsections. We assume that the memristors have k-bit resolution, which allows us to program them with one of 2^k distinct conductance values $\{0, 1, \ldots, 2^k - 1\}$. We first focus on k-bit addition and subtraction operations, where both the operands and also the final results are assumed to be k-bit quantities.

A. Addition and Subtraction

Here we store k-bit operands as conductance values in the memristors, and the results are obtained as currents (mapped to k-bit values) along the columns. Some of the possible operations are explained below.

1) Addition of two scalars: We can compute the sum of two k-bit scalars a_1 and a_2, where $0 \leq a_1, a_2 < 2^k$, as:

$$s \quad = \quad a_1 + a_2 \tag{2}$$

To carry out this operation, we do the following:

i) Set $g_{11} = a_1$, $g_{21} = a_2$.
ii) Set $V_1 = V_2 = 1$, $V_3 = V_4 = \cdots = V_n = 0$.

The result is obtained as the current $I_1 = g_{11} + g_{21} = a_1 + a_2$, which is treated as the analog equivalent of a k-bit value.

2) Addition of n scalars: The concept can be extended to compute the sum of n k-bit scalars as:

$$s \quad = \quad a_1 + a_2 + \cdots + a_n \tag{3}$$

where $0 \leq a_1, a_2, \ldots, a_n < 2^k$. For this, we do the following:

i) Set $g_{11} = a_1,\ g_{21} = a_2, \ldots,\ g_{n1} = a_n$.
ii) Set $V_1 = V_2 = \cdots = V_n = 1$.

The sum is again obtained as current value in the first column: $I_1 = g_{11} + g_{21} + \cdots g_{n1} = a_1 + a_2 + \cdots + a_n$.

3) Handling negative operands and subtraction: To handle negative operands as proposed in literature [4], [17], [18], we replace every column by a pair of columns. Fig. 2. The i^{th} column with conductance values $(g_{1i}, g_{2i}, \ldots, g_{ni})$ is replaced by a pair of columns with conductance values $(g^+_{1i}, g^+_{2i}, \ldots, g^+_{ni})$ and $(g^-_{1i}, g^-_{2i}, \ldots, g^-_{ni})$ respectively, with column currents denoted as I^+_i and I^-_i. The final current is obtained as:

$$I_i \quad = \quad I^+_i - I^-_i \tag{4}$$

This way we can use negative operands as well as carry out subtraction. For instance, to carry out the scalar subtraction $s = a_1 - a_2$, we do the following

i) Set $g^+_{11} = a_1,\ \ g^-_{11} = a_2$.
ii) Set $V_1 = 1$, $V_2 = V_3 = V_4 = \cdots = V_n = 0$.

We get $I_1 = g^+_{11} - g^-_{11} = a_1 - a_2$.

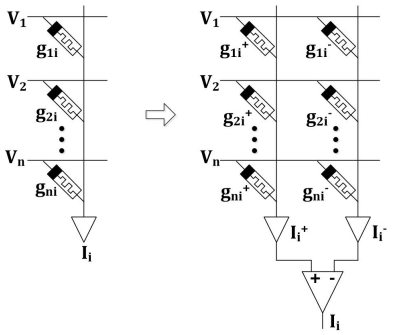


Fig. 2. Handling negative operands

4) Addition of two vectors: We can add two vectors $A = (a_1, a_2, \ldots, a_m)$ and $B = (b_1, b_2, \ldots, b_m)$ and produce the sum vector $C = (c_1, c_2, \ldots, c_m)$ as:

$$c_i \quad = \quad \sum_{i=1}^{m} (a_i + b_i) \tag{5}$$

To carry out this operation, we do the following:

i) Set $g_{11} = a_1,\ g_{12} = a_2, \ldots,\ g_{1m} = a_m$.
ii) Set $g_{21} = b_1,\ g_{22} = b_2, \ldots,\ g_{2m} = b_m$.
iii) Set $V_1 = V_2 = 1$, and $V_3 = \cdots = V_n = 0$.

The sum is obtained as current values in each of the m columns: $I_i = g_{1i} + g_{2i} = a_i + b_i$, for $i = 1, 2, \ldots, m$.

B. Scalar and Vector Multiplication

We show how multiplication can be carried out on the crossbar, where some of the operands can be scalar while some can be a vector as well.

1) Normal vector-matrix multiplication: The conventional VMM operation can be directly computed as:

$$I_{1\times m} \quad = \quad V_{1\times n} \times G_{n\times m} \tag{6}$$

where the G matrix is fed as conductance values in the memristors, and the vector V as voltages along the rows.

2) Multiplication of two scalars: Here, k-bit digital-to-analog converters (DACs) are used to generate 2^k-valued voltage inputs $\{V_1, V_2, \ldots, V_n\}$ for the crossbar. Each D_i represents a k-bit number in the range 0 to $2^k - 1$. We can compute the scalar product $p = a \times b$, where $0 \leq a, b, p < 2^k$. One of the operands is stored as conductance value in the crossbar, while the other is applied as a k-bit number to the input of the DAC. The steps are:

i) Set $D_1 = a$, $g_{11} = b$.
ii) Set $D_2 = D_3 = \cdots = D_n = 0$.

The final product is obtained as $I_1 = V_1.g_{11}$.

3) Multiplication of a vector by a scalar: Here, we compute the product of a scalar a and a vector $B = (b_1, b_2, \ldots, b_m)$, to generate the product $P = (p_1, p_2, \ldots, p_n)$:

$$p_i \quad = \quad a \times b_i \tag{7}$$

for $i = 1, 2, \ldots, m$. The required steps are as follows:

i) Set $D_1 = a$, $D_2 = D_3 = \cdots = D_n = 0$.
ii) Set $g_{11} = b_1$; $g_{12} = b_2$; ...; $g_{1m} = b_m$.

The final product is: $I_1 = a.b_1, I_2 = a.b_2, \ldots, I_m = a.b_m$.

4) Inner product of two vectors: We compute the inner product of vectors $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ as:

$$p \quad = \quad \sum_{i=1}^{n} (a_i \times b_i) \tag{8}$$

The vector A is applied to the inputs of the DACs, and B as conductance values in the first column. The steps are as:

i) Set $D_1 = a_1, D_2 = a_2, \ldots, D_n = a_n$
ii) Set $g_{11} = b_1, g_{21} = b_2, \ldots, g_{n1} = b_n$.

The product is obtained as the current I_1 in the first column.

IV. Enhancing Bit Resolution for Computation

As discussed, k-bit data can be stored as conductance values in memristors, or applied as inputs to k-bit DACs. Although DACs with large k (say, > 32) can be designed, it is difficult to fabricate memristors with enhanced resolution (typically, $k < 8$). We present a technique to enhance the resolution of the data operands to some multiple of k.

The approach is illustrated in Fig. 3 for a single crossbar column. In Fig. 3(a), we show the i^{th} column of the original crossbar, which is replaced by p columns as shown in Fig. 3(b) to provide kp bits of resolution. The opamp computes the

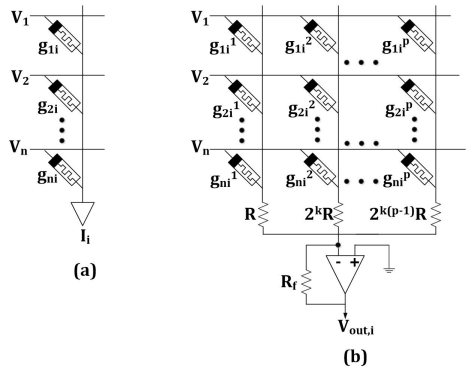


Fig. 3. Increasing the bit-storage resolution in crossbar

(binary) weighted sum of the currents produced in the p columns. The effective weights will be:

$$\begin{aligned} g_{1i} &\equiv 2^{k(p-1)}g^1_{1i} + 2^{k(p-2)}g^2_{1i} + \cdots g^p_{1i} \\ g_{2i} &\equiv 2^{k(p-1)}g^1_{2i} + 2^{k(p-2)}g^2_{2i} + \cdots g^p_{2i}, \text{ and so on.} \end{aligned}$$

Thus, each weight g_{ki} is replaced by a kp-bit weight $G_{ki} = (g^1_{ki}, g^2_{ki}, \ldots, g^p_{ki})$. It may be noted that

$$V_{out,i} = \beta\left(V_1G_{1i} + V_2G_{2i} + \cdots + V_nG_{ni}\right) \quad (9)$$

where β is a constant.

A. Addition with Enhanced Bit Resolution

Referring to Fig. 3(b), we illustrate the process of adding of two kp-bit numbers. The numbers to be added are denoted as: $A = \left(a^1, a^2, \ldots, a^p\right)$ and $B = \left(b^1, b^2, \ldots, b^p\right)$, where a^i and b^i are k-bit quantities.

We initialize the memristors in the crossbar as follows:

i) Set $g^1_{11} = a^1$, $g^2_{11} = a^2$, …, $g^p_{11} = a^p$.
ii) Set $g^1_{21} = b^1$, $g^2_{21} = b^2$, …, $g^p_{21} = b^p$.
iii) Set $V_1 = V_2 = 1$, $V_3 = \cdots = V_n = 0$.

The final output is obtained as:

$$\begin{aligned} V_{out,1} &= 2^{k(p-1)}(g^1_{1i} + g^1_{2i}) + 2^{k(p-2)}(g^2_{1i} + g^2_{2i}) \\ &\quad + \cdots + (g^p_{1i} + g^p_{2i}) \end{aligned}$$

which is the sum of the two kp-bit numbers A and B.

B. Multiplication with Enhanced Bit Resolution

Again referring to Fig. 3(b), we assume that the DACs have kp-bit inputs. We feed the first number to the DAC in the first row, and set the second number as conductance values in the first row. In other words, we do the following:

i) Set $D_1 = A$, $D_2 = D_3 = \cdots = D_n = 0$.
ii) Set $g^1_{11} = b^1$, $g^2_{11} = b^2$, …, $g^p_{11} = b^p$.

We get the final output as:

$$\begin{aligned} V_{out,1} &= 2^{k(p-1)}(A * g^1_{1i}) + 2^{k(p-2)}(A * g^2_{1i}) \\ &\quad + \cdots + (A * g^p_{1i}) \\ &= A\left(2^{k(p-1)}g^1_{1i} + 2^{k(p-2)}g^2_{1i} + \cdots\right) \end{aligned}$$

which is the product of the two kp-bit numbers A and B.

V. Analysis of Fault Tolerance Capability

In a previous work [19], the ideal accuracy of a crossbar was compared with the accuracy in the presence of faulty memristors. A similar approach is followed for analyzing the fault tolerance capability of the proposed architecture.

In [19], for analyzing the fault tolerance capability of crossbar, offline training approach is used. In offline training, the network is trained on a host system and the final trained weights are downloaded to the crossbar for evaluation. Initially the network is trained by considering the crossbar as ideal (i.e., zero fault), and the network is evaluated by downloading the learnt weights on the crossbar. In the next step, faults are injected into the crossbar and the network is evaluated again. If the accuracy of ideal case and faulty case does not differ by more than 1%, then the percentage of faults is increased. The maximum percentage of faults that can be tolerated is found out, with respect to the threshold of 1%.

A comparative analysis is depicted in Table I for p-bit resolution ($2 \leq p \leq 5$), for two of the benchmark datasets MNIST [20] and CIFAR-10 [21]. The column labeled *Simple* corresponds to a crossbar of p-bit memristors, where a p-bit weight is stored in a single memristor. The column *Proposed* is for the proposed architecture with enhanced bit resolution with $k = 1$ (each memristor with 1-bit resolution), where p columns are used to store a p-bit weight.

TABLE I
Percentage accuracy for ideal crossbars.

No. of Bits (p)	MNIST [20]		CIFAR-10 [21]	
	Simple [19]	Proposed	Simple [19]	Proposed
2-bit	77.92	77.41	72.75	72.85
3-bit	79.06	80.13	74.29	74.68
4-bit	80.23	80.65	75.83	75.97
5-bit	81.40	82.04	76.54	76.12

It can be observed that the proposed architecture achieves a similar accuracy as compared to the *Simple* approach, where an ideal crossbar with p-bit resolution is assumed.

Fig. 4 shows the average variation in accuracy for multi-bit memristor for MNIST dataset for *Simple* and *Proposed* crossbar architectures with various % of faults. If the accuracy of the crossbar lies within the range marked by the red lines (less than $\pm 1\%$ variation), it is considered as tolerable. Plots for four different values of p are shown (2, 3, 4 and 5).

From the plots it can be observed that the proposed approach shows higher fault tolerance as compared to a simple crossbar where each weight is stored in a single memristor. This is because in the latter case, in the presence of fault the weight will become either 0 (stuck-at-low) or $2^k - 1$ (stuck-at-high). However, in the proposed approach if one memristor is faulty, only a part of the weight gets affected as weights are stored across multiple memristors.

VI. Case Studies for Approximate Computing

Referring to the typical values quoted in [22], the average latency for VMM is in the range 1-5μsec, including the delay of the DACs. Also, assuming an average row voltage of 2.5V, average resistance of 50kΩ, and average latency of 2.5μsec,

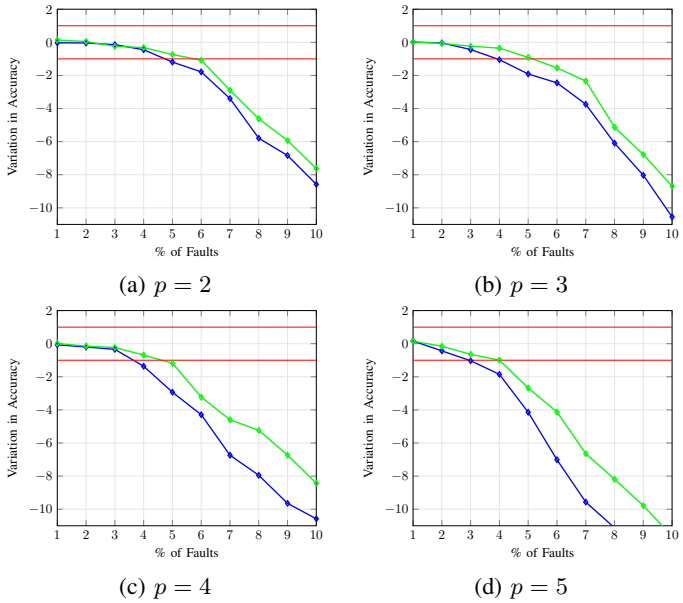


Fig. 4. Variation in accuracy for *Simple* (blue line) and *Proposed* (green line.

the energy consumption of a memristor can be estimated as 0.31 nJ. For neuromorphic operations, most of the latency and energy are contributed in the DACs and ADCs (where required). However, almost all the operations mentioned before can be performed in a single cycle.

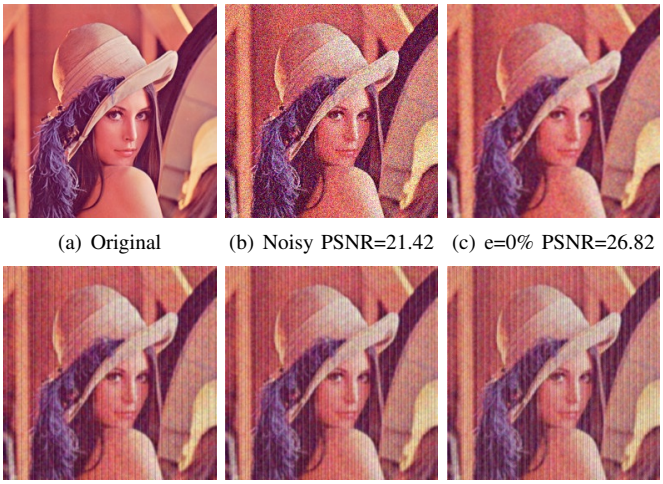
We now present some case studies where AC can be used, and illustrate the effectiveness of the proposed approach.

A. Case-Study: Approximate Computing

In order to show the applicability of our approach, we have performed a case-study, where we have chosen two error-tolerant applications from different fields. We show that even in the presence of faulty devices, using our proposed approach we can still achieve satisfying results for the given applications. As fault models we have used the most common ones, viz. stuck-at-1 and stuck-at-0 faults. We have assumed that e percent of the cells are faulty and they are randomly at stuck-at-1 or stuck-at-0 with the same probability of 50%.

1) Application 1: Image Smoothing:

a) ***Implementation Details****:* We have simulated the application of a Gaussian filter for image smoothing using a memristive crossbar. Here, we have simulated memristors with 4-bit resolution (i.e. each can store 4 bits). For the Gaussian filter, we have used the filter proposed in [23] and applied it to the well-known Lena benchmark. Here every pixel consists of three channels with each having a value between 0 and 255. Since we assume memristors with 4-bit resolution, two memristors are needed to store each 8-bit value. The Gaussian filter consists of 25 values and is convolved across the image. Here, two memristors in each of 25 wordlines are used to store the respective values. Consequently, each convolution can be computed using the presented method for the computation of an inner product of two vectors. As the values of the Gaussian filter are constant, we can parallelize this convolution operation, by using multiple bitlines within these wordlines to store adjacent values of the image. Here, we assume 16-bit wordsize. Consequently, as each value needs two memristors we can compute eight values of the convolution in parallel.



(a) Original (b) Noisy PSNR=21.42 (c) e=0% PSNR=26.82

(d) e=5% PSNR=24.06 (e) e=10% PSNR=23.66 (f) e=20% PSNR=22.73

Fig. 5. Different PSNR values after Gaussian filter.

b) ***Results****:* The results are shown in Fig. 5. Fig. 5(a) shows the original Lena image to which we have applied random noise in Fig. 5(b). This results in a PSNR of 21.42. We have then applied the Gaussian filter with different values for $e \in \{0\%, 5\%, 10\%, 20\%\}$, which can be seen in Fig. 5(c)-(f), respectively. It can be seen that even with $e = 20\%$ faulty cells, the PSNR can be enhanced using Gaussian filter compared to the disturbed image to which the filter has been applied. It is interesting to see that the simulated errors of the cells cause a systematic disturbance in the final output of the algorithm (vertical lines in Fig. 5(d)-(f)). This is due to the fact that we assume that the same cells are reused after every computation in order to keep the overhead of the used area low.

2) Application 2: k-Nearest Neighbor Classification:

a) ***Implementation Details****:* In order to implement our error model, we have used the repository [24] as basis. This repository comes with the Iris plant dataset [25] that consists of 150 samples assigned to one of three classes. We divide these samples into a training dataset with 120 images and a test dataset that contains 30 images randomly. In order to allow for in-memory computation, we have converted the coordinates of the data points, which are given as four floating point numbers, to four unsigned 16-bit fixed point numbers, with 12 bits used for the decimal part. Since we assume memristors with 4-bit resolution, four cells are needed to store each dimension of the coordinate for a data point. We assume that all points of the dataset are stored within the crossbar. Consequently, some are distorted due to the stuck-at-1 and stuck-at-0 faults. To compute the distance between two data points, we assume that the presented method for the subtraction of two vectors is used. This way, the difference between two values can be computed. Then, in order to compute the square of such a difference, we assume that the difference is first stored to random cells within a wordline and then the difference is applied as voltage to the wordline.

Here, we assume that the memristors used are faulty with the probability e. Finally, the squares have to be summed. Again, here we assume that random cells are used and consequently, the memristors in which the squared differences are stored are faulty with the probability e and the placement of the faulty cells. Note that we can skip computing the root of this sum, since $\sum_i a^2 < \sum_i b^2 \implies \sqrt{\sum_i a^2} < \sqrt{\sum_i b^2}$ and only this relation is important to identify the k nearest neighbors. In our experiments, we have used $k = 5$.

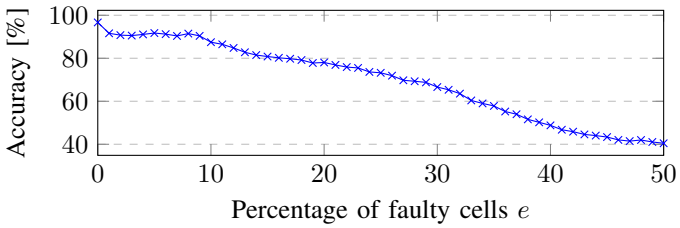


Fig. 6. Evaluation of the accuracy of the kNN classifier with faulty cells.

b) ***Results****:* In our experiments, we have applied the kNN classifier for $e \in [0\%, .., 50\%]$. We have applied it 1000 times to the dataset. The results can be seen in Fig. 6, which shows the mean accuracy of the test dataset over all runs for different values of e. Initially, if no cells are faulty, the accuracy is at 96.67%. We can see that the accuracy of the classifier stays almost above 80% on average, if the percentage of faulty cells e stays above 17%. However, we can see that even with 50% faulty cells, the results are still classified with more than 40% accuracy on average, which is still significantly better than 33% which would be guessing.

It is worth noting that the quality of the results does vary depending on where the faulty cells are. For $e = 10\%$, the highest accuracy obtained is 96.67% which is same as the result when no cells are faulty. The lowest accuracy is 73.33% resulting in a difference of upto 23%, depending on e. However, finding erroneous cells and adjusting the placement of the operands depending on their position and their respective faults is beyond the scope of this paper.

VII. Conclusion

In this paper we show how the memristor crossbar can be used for performing various scalar and vector arithmetic operations. We also present a method to enhance the resolution of data operands and show how different operations can be performed with higher resolution. We also analyze the fault tolerant capability of the approach. We further show how these operations can be used in AC applications where a compromise is possible with respect to computational accuracy. We have considered two error-tolerant applications, viz. image smoothing and k-nearest neighbor classification, and analyzed their performance on memristor based crossbar.

References

[1] L. Chua, "Memristor – the missing circuit element," *IEEE Trans. on Circuit Theory*, vol. CT-18, no. 5, pp. 507–519, 1971.

[2] L. O. Chua and M. K. Sung, "Memristive devices and systems," *Proceedings of the IEEE*, vol. 64, pp. 209–223, February 1976.

[3] K. Akarvardar and H. S. P. Wong, "Ultralow voltage crossbar nonvolatile memory based on energy-reversible NEM switches," *IEEE Electron Device Letters*, vol. 30, no. 6, pp. 626–628, 2009.

[4] M. Prezioso *et al.*, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[5] H. Wang, X. Yan, S. Wang, and N. Lu, "High-stability memristive devices based on pd conductive filaments and its applications in neuromorphic computing," *ACS Applied Materials & Interfaces*, vol. 13, no. 15, pp. 17844–17851, 2021.

[6] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, no. 1, p. 13, 2013.

[7] R. Gharpinde, P. L. Thangkhiew, K. Datta, and I. Sengupta, "A scalable in-memory logic synthesis approach using memristor crossbar," *IEEE Trans. on VLSI Syst.*, vol. 26, pp. 355–366, Feb 2018.

[8] L. Xia *et al.*, "Technological exploration of RRAM crossbar array for matrix-vector multiplication," *Journal of Computer Science and Technology*, vol. 31, pp. 3–19, Jan 2016.

[9] A. Shafiee, A. Nag, N. Muralimanohar, and R. Balasubramonian, "ISAAC: A convolution neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[10] M. Hu *et al.*, "Memristor-based analog computation and neural network classification with a dot product engine," *Advanced Materials*, vol. 30, no. 9, p. 1705914, 2018.

[11] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *DAC Design Automation Conference*, pp. 820–825, 2012.

[12] A. Chandrasekharan, D. Große, and R. Drechsler, "ProACt: A processor for high performance on-demand approximate computing," in *Proceedings of the on Great Lakes Symposium on VLSI*, p. 463–466, 2017.

[13] M. Yan *et al.*, "kNN-CAM: A k-nearest neighbors-based configurable approximate floating point multiplier," in *Intl. Symp. on Quality Electronic Design*, pp. 1–7, 2019.

[14] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, "Approximation-aware rewriting of AIGs for error tolerant applications," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2016.

[15] V. Mrazek, Z. Vasicek, L. Sekanina, M. A. Hanif, and M. Shafique, "ALWANN: Automatic layer-wise approximation of deep neural network accelerators without retraining," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019.

[16] A. John, S. Ullah, A. Kumar, B. Cardiff, and D. John, "An approximate binary classifier for data integrity assessment in IoT sensors," in *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 1–4, 2020.

[17] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang, "Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar," in *Design, Automation and Test in Europe (DATE)*, pp. 19–24, IEEE, 2017.

[18] R. Hasan, T. M. Taha, and C. Yakopcic, "On-chip training of memristor crossbar based multi-layer neural networks," *Microelectronics Journal*, vol. 66, pp. 31–40, 2017.

[19] D. N. Yadav, K. Datta, and I. Sengupta, "Analyzing fault tolerance behaviour in memristor-based crossbar for neuromorphic applications," in *2020 IEEE International Test Conference India*, pp. 1–9, IEEE, 2020.

[20] Y. LeCun, "The MNIST database of handwritten digits," *http://yann.lecun.com/exdb/mnist/*, 1998.

[21] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (Canadian Institute for Advanced Research)," vol. 5, p. 4, 2009.

[22] A. Amirsoleimani *et al.*, "In-memory vector-matrix multiplication in monolithic complementary metal–oxide–semiconductor-memristor integrated circuits: Design choices, challenges, and perspectives," *Advanced Intelligent Systems*, vol. 2, p. 2000115, 2020.

[23] M. S. Lau, K.-V. Ling, and Y.-C. Chu, "Energy-aware probabilistic multiplier: Design and analysis," in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, p. 281–290, 2009.

[24] J. N. Carvalho, "KNN k-nearest-neighbors in C++." https://github.com/joaocarvalhoopen/KNN__K_Nearest_Neighbors_in_C_Plus_Plus, 2019.

[25] D. Dua and C. Graff, "UCI machine learning repository," 2017.