Input-Aware Flow-Based In-Memory Computing

Suraj Singireddy*, Muhammad Rashedul Haq Rashed†, Sven Thijssen†, Rickard Ewetz† and Sumit Kumar Jha‡
*Computer Science Department, University of Texas at San Antonio, San Antonio, USA
†Department of Electrical and Computer Engineering, University of Central Florida, Orlando, USA
‡Computer Science Department, Florida International University, Miami, USA
suraj.singireddy@utsa.edu, {muhammad.rashed, sven.thijssen, rickard.ewetz}@ucf.edu, jha@cs.fiu.edu

***Abstract*—In-memory computing using nanoscale crossbar arrays is a promising solution strategy to overcome the limitations of the von Neumann architecture. Flow-based computing is an emerging in-memory computing paradigm for evaluating Boolean logic using the natural flow of electrical currents. Previous studies on flow-based computing have focused on synthesizing crossbar designs with small dimensions to improve various performance metrics. In this paper, we observe that the latency and energy of evaluating a Boolean input vector is dependent on the state of the crossbar design (or the previous input vector). To take advantage of this observation, we propose the REORDER framework that reorders the sequence of input vectors to improve performance. The reordering reduces the overall number of WRITE operations to the non-volatile memory devices, which has a first-order impact on the overall performance of flow-based computing systems. The *optimal* input sequence can be obtained by formulating and solving a traveling salesman problem (TSP). The REORDER framework leverages a heuristic solution to balance pre-processing overhead with reduction in device switching. We evaluate the REORDER framework on image processing applications that allow input vector reordering. Compared with a naïve input sequence, the framework improves time and energy efficiency by 78% and 69% respectively for image filtering and by 94% and 72% respectively for feature extraction.**

I. Introduction

Complementary metal-oxide-semiconductor (CMOS) technology, which has largely driven the advancement of energy efficiency and throughput in digital processors for the past 50 years, is rapidly approaching its physical limits [1]. It is estimated that, within the decade, transistors with a gate length of 5 nm will be manufactured - beyond this limit, quantum-mechanical effects can interfere, resulting in current leakage and energy loss [2]. As a result, the race to scale CMOS has nearly come to an end, forcing us to investigate alternate devices, architectures, and computing paradigms with beyond-Moore potential. Moreover, a primary constraint in modern computing architectures is the von Neumann bottleneck - the cost of communication between separate memory and processing units. The concept of in-memory computing, in which memory and computation are performed on the same device, resolves the von Neumann bottleneck by removing the cost of memory access altogether [3].

Hardware realizations of in-memory computing require a memory device as well as a fabric for performing computation. In 1971, Leon Chua proposed the existence of the memristor [4], which was subsequently manufactured in 2008 [5]. Dense crossbar arrays of memristive devices are a strong candidate for in-memory computing due to their ability to store data with close to zero leakage power [6]. Several in-memory computing paradigms based on memristor crossbars have been proposed, such as memristor-aided logic (MAGIC) [7], material implication (IMPLY) [8], analogue matrix-vector multiplication accelerators [9], and flow-based computing. Flow-based computing is an emerging in-memory computing paradigm which is particularly promising for evaluating Boolean logic [10]. Memory is loaded onto the crossbar in a precise pattern which is designed to map the flow of current into the desired Boolean computation. Then, a current is applied to the input nanowire and the crossbar output is measured at one or more output nanowires. This approach is particularly effective in settings where a computation is executed many times on different inputs since the design pattern can be used repeatedly.

Research on flow-based computing has been focused on automatically synthesizing Boolean functions into crossbar designs. The objective is to minimize the dimensions of the resulting crossbars, which translates into performance and area improvements. Early synthesis tools were driven by model counting [11] and Boolean satisfiability [12]. More recent studies perform synthesis by leveraging Binary Decision Diagrams (BDDs). An optimal BDD embedding technique was proposed in [10]. With no further gains to be expected from reducing crossbar size, alternative approaches to improving performance must be explored.

The predominance of the von Neumann architecture has motivated efforts to optimize performance and energy efficiency of von Neumann systems at all levels, including the physical implementation, computer architecture, circuit and logic networks, compilers, and operating systems. However, the fundamental limits of CMOS technology have made it difficult to improve the efficiency of such systems. It is critical that we leverage the understanding gained by optimizing traditional computing devices in the development of successor technologies. In CMOS-based systems, power efficiency is achieved by minimizing switching activity, since power is not dissipated if the circuit is not switching [13]. We observe that the switching activity induced by evaluating a Boolean input vector is largely dependent on the state of the crossbar, which is defined by the previous input vector. Hence, there exists an opportunity to improve performance by intelligently reordering input vectors.

In this paper, we propose a framework called REORDER that improves performance through input-aware flow-based computing. The framework minimizes the number of memristor

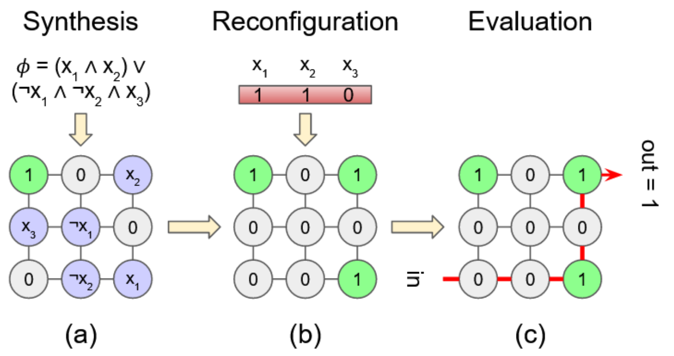

Fig. 1. (a) Crossbar design obtained from synthesis. (b) Reconfiguration of hardware with respect to an input vector (x_1,x_2,x_3)= (1,1,0). (c) Evaluation of ϕ using $\mathcal{D}$.

switches by reordering the sequence of the input vectors. Memristors are switched using WRITE operations, which have a first-order impact on the overall performance. We observe that the number of required device switches between each pair of input vectors can be stored in a graph. Finding the *optimal* input sequence that minimizes the amount of device switching is equivalent to solving the traveling salesman problem (TSP) in this graph. The REORDER framework utilizes Gray code ordering to efficiently find low-cost Hamiltonian paths in this setting, outperforming general-purpose TSP solvers, heuristics, and approximation algorithms. Additionally, we show that there are natural opportunities to reorder input vectors within important image processing applications. The framework is evaluated using six image filtering techniques on the CIFAR-10 dataset as well as feature extraction on MNIST. The experimental results demonstrate latency and energy improvements of 78% and 69% respectively for image filtering and improvements of 94% and 72% respectively for feature extraction.

The remainder of this paper is organized as follows: background is provided in Section II. The concept of input aware flow-based computing and problem formulation are discussed in Section III. The REORDER framework is presented in Section IV. Our method for performing image filtering and feature extraction using flow-based computing is described in Section V. The experimental evaluation is given in Section VI. The paper is concluded in Section VII.

II. Background

A. Memristor Crossbar Arrays

A memristor crossbar array consists of two layers of perpendicular metal nanowires. Within each intersection point, there is a memristor sandwiched between the metal wires. A memristor is a two-terminal device with programmable resistance. The memristor acts as a switch that is turned on (off) when the resistance is programmed to be low (high). The resistance state of a memristor can be updated via a WRITE operation, in which a current is simultaneously applied to its incident horizontal and vertical nanowires. This causes a voltage drop of the desired magnitude across the target memristor and half the voltage drop across memristors in the same row or column [14].

While resistive memory has several benefits - it is non-volatile and achieves both low write energy and high memory density - the existence of sneak-paths causes unwanted flows of current in nanoscale crossbar arrays which prevent the accurate reading of memristor states and increase energy consumption [15]. These normally parasitic currents are instead used to perform efficient computation within flow-based computing [16].

B. Flow-Based Computing

Flow-based computation consists of three phases: synthesis, reconfiguration, and evaluation. In the one-time synthesis phase, a Boolean function ϕ is synthesized into a crossbar design $\mathcal{D}$. Each memristor in a crossbar design has been assigned a constant, Boolean variable, or complemented Boolean variable, which is shown in Figure 1(a). The crossbar realizes the Boolean formula $\phi = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3)$. A significant benefit of flow-based computing is that, after a single costly design synthesis phase, the same crossbar design can be used to efficiently evaluate many input vectors. Each crossbar design corresponds to a Boolean formula, and each input vector consists of an assignment of values to the Boolean variables.

In the reconfiguration phase, the memristors in the crossbar are programmed to be on/off with respect to a Boolean input vector, which is shown in Figure 1(b). The figure shows the reconfiguration with respect to $x_1 = 1, x_2 = 1, x_3 = 0$. To evaluate the instance, a voltage is applied to the input wordline. Current flows between any nanowires which are connected by a memristor in the low-resistance state, denoted by 1. Finally, since the current reaches the output nanowire through a sneak-path of low-resistance memristors, the crossbar produces the output of 1. Otherwise, the function evaluates to 0.

III. Input-Aware Flow-Based Computing

A. Motivation

A crossbar design $\mathcal{D}$ is used to evaluate the corresponding Boolean function ϕ for many different input vectors. The reconfiguration step dominates the overall latency and power consumption because it relies on expensive WRITE operations, whereas the evaluation phase requires only a single READ operation. The number of WRITE operations in the reconfiguration phase is dependent on the state of the crossbar as defined by the previous input vector.

Two different input sequences of the same Boolean input vectors are shown in (a) and (b) of Figure 2. Both input sequences are processed by the the crossbar design in Figure 2(c). The intermediate crossbar states for the input sequence I_1 is shown in Figure 2(d). It can be observed that 2 devices are switched between input vector v_0 and v_1 in Figure 2(d). The complete sequence I_1 results in a total of 10 WRITE operations. In contrast, the input vector sequence I_2 results only in 7 WRITE operations. Hence, there exists an opportunity to minimize the total number of WRITE operations by reordering the input vectors. We also note that there exists natural opportunities

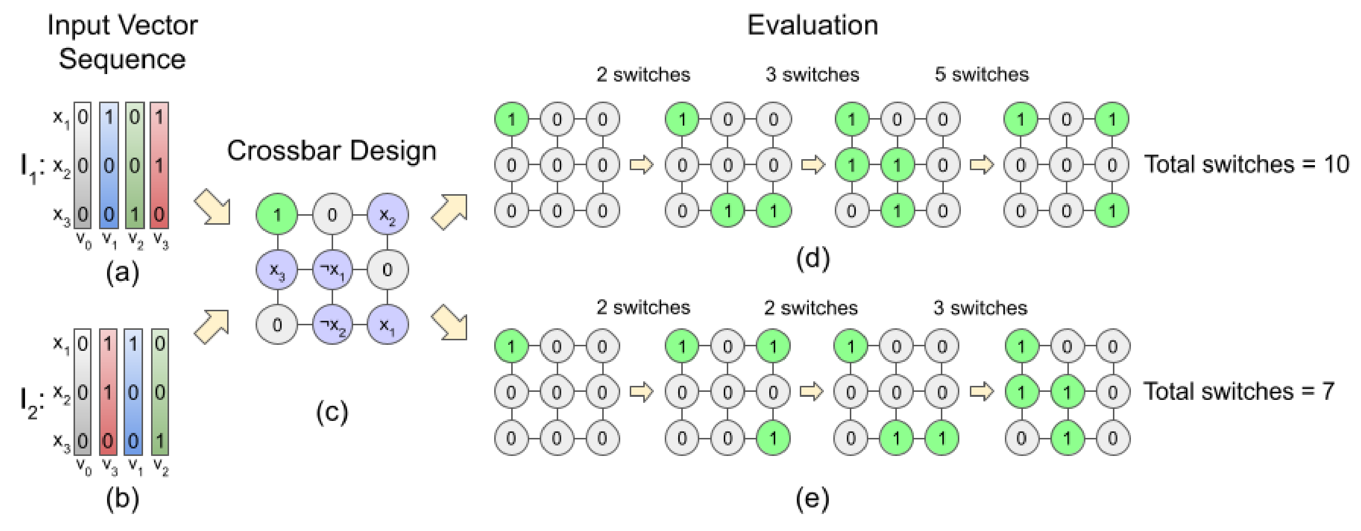

Fig. 2. (a) Input vector sequence $I_1 = \{v_0, v_1, v_2, v_3\}$. (b) Input vector sequence $I_2 = \{v_0, v_3, v_1, v_2\}$. (c) Crossbar design $\mathcal{D}$. (d) Intermediate state of the crossbar after each input vector in (a). (e) Intermediate state of the crossbar after each input vector in (b).

for input vector reordering within many image processing applications; this discussion is continued in Section V.

B. Problem Definition

The goal of this paper is to minimize the time and energy consumption for a flow-based computing system to execute a sequence of input vectors. At test time, the system is given a sequence of inputs to be evaluated. Between each evaluation, the new input vector must be loaded onto the crossbar by reconfiguring the resistance states of the memristors. Given a crossbar design representing some target function and a input vector sequence of problem instances to evaluate, we aim to evaluate the sequence using the minimum time and energy possible.

The input is a crossbar design $\mathcal{D}$ and n input vectors $I_{init} = \{v_0, \ldots, v_n\}$. The initial state of the crossbar design is defined by a pseudo input vector v_0 that cannot be reordered. The input vectors v_1 to v_n are from an application that allows the order of the input vectors to be rearranged. We approach this problem by seeking the input sequence I that minimizes the device switching activity.

The **Input Vector Ordering Problem** can be formulated as finding the input sequence I_{opt} that minimizes the number of WRITE operations:

$$I_{opt} = \arg\min_{I} \sum_{k=0}^{n-1} d(I[k], I[k+1]) \tag{1}$$

where $I[k]$ denotes the k^{th} input vector in the sequence I. $d(I[k], I[k+1])$ denotes the number of memristors that must be switched between evaluating the input vectors $I[k]$ and $I[k+1]$.

IV. REORDER Framework

In this section, we present the REORDER framework to solve the input vector ordering problem. The reordering of the input vectors is based on an analogy between reordering and the traveling salesman problem (TSP), which is outlined in Section IV-A. The methodology of the framework is presented in Section IV-B.

A. Analogy between Input Vector Reordering and TSP

It has been observed [17], [18] that the problem of reordering input vector sequences to minimize power dissipation in CMOS circuits can be recast as an instance of the traveling salesman problem. We modify this formulation to minimize switching activity in flow-based computing systems as follows: Let $G = (V, E)$ be a graph where the vertices V represent input vectors and the edges E represent the number of required WRITE operations between pairs of sequential inputs. In particular, the distance between each pair of input vectors (v_i, v_j) represents the cost of reprogramming the crossbar array to evaluate v_j given that it is currently configured to evaluate v_i. These pairwise distances can be computed approximately using the Hamming distance between input vectors; although the exact switching cost depends upon the state of the crossbar array [14], it is sufficiently accurate to assume constant values for time and power efficiency. Consequently, determining the input sequence that requires the fewest WRITE operations is equivalent to finding the shortest Hamiltonian path in the graph, which is the well-known traveling salesman problem (TSP).

In input vector reordering, the energy saved by executing a reordered input vector sequence must outweigh the cost of computing the reordering; however, solving the TSP is famously NP-complete. Therefore, rather than computing the exact minimum-cost path, the REORDER framework utilizes efficient TSP heuristics to balance the energy saved from the reordering with the overhead of computing the optimized input sequence.

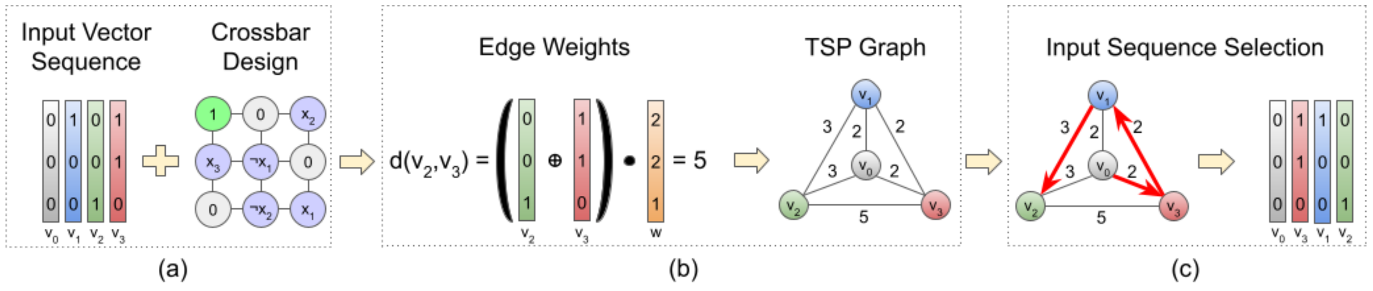

Fig. 3. (a) Sequence of input vectors and a crossbar design $\mathcal{D}$. (b) TSP graph construction. (c) Reordering input sequence based on shortest path.

B. Methodology

The input to the REORDER framework is the crossbar design $\mathcal{D}$ and the default sequence of input vectors I_{init}. The output is an optimized input order sequence I_{out}. The framework consists of a TSP graph construction step and input sequence selection step. The flow of the framework is illustrated with an example in Figure 3.

1) TSP Graph Construction: The input to the TSP graph construction is the default input sequence I_{init} and the crossbar design $\mathcal{D}$, which is shown in Figure 3(a). In this step, we construct the TSP graph that captures the number of required WRITE operations between each pair of input vectors. The first step is to count the number of times each Boolean variable x_k (including complemented variables x_k) occurs in the crossbar design. We denote the result as the frequency vector f. The frequency vector $f = [2,2,1]^T$ for the crossbar design in Figure 3(a) is shown in Figure 3(b). Using the frequency vector, the number of required WRITE operations $d(v_i, v_j)$ between two sequential inputs v_i and v_j can be computed as the Hamming distance between the input vectors weighted by f:

$$d(v_i, v_j) = (v_i \oplus v_j)^T f \tag{2}$$

where $\oplus$ is the bitwise exclusive or function. Based on Eq (2), we compute $d(v_2, v_3) = 5$ in Figure 3(b).

Now we are ready to formulate the TSP graph $G = (V, E)$. The graph is a complete graph with a vertex for each input vector $v_0, \ldots v_n$. The input vector v_0 is a pseudo input vector that is used to define the initial state of the crossbar. There is an edge (i, j) between each pair of nodes and its weight $w_{i,j}$ is set to $d(v_i, v_j)$ using Eq (2), which is shown to the right of Figure 3(b).

2) Input Sequence Selection: Each Hamiltonian path in the graph corresponds to an input vector sequence. To ensure that the initial configuration remains at the beginning of the sequence, we fix the starting node of each path to v_0. The optimal Hamiltonian path for the TSP graph in Figure 3(b) is shown to the left in Figure 3(c). The resulting sequence of input vectors is shown to the right in Figure 3(c). The input vector sequence is simply obtained from the order of the traversed nodes.

As the TSP problem is an NP-complete problem, it is expensive to solve exactly. Therefore, it is important to carefully balance the performance benefits from reordering with the cost of computing the new input vector sequence. In our experimental evaluation we compare Concorde [19], an exact TSP solver, and several heuristic methods, including the Lin-Kernighan algorithm [20], the Christofides algorithm [21], and an adaptation of Gray code ordering [22]. We observe that Gray code ordering gives the best results; we describe the details of the method below.

Although the general TSP is NP-complete, variants of the TSP problem where the distance between two nodes is equal to some metric (e.g. Euclidean distance) admit fast and accurate heuristic solutions [23]. We observe that the number of WRITE operations required to reprogram a flow-based computing crossbar instance can be computed as the weighted Hamming distance between the previous input vector v_i and the next input vector to be evaluated v_j. Thus, input vector reordering is a special case of the TSP where the distances constitute a weighted Hamming space.

Gray code ordering can be used to solve the TSP for binary Hamming spaces [18]; in other words, problems for which the distance between any two nodes is equal to the Hamming distance of the corresponding bit vectors. In this setting, the TSP can be reduced to the problem of finding a minimal-change order of the binary code given by the input vector sequence.

In an n-dimensional binary Hamming space, any n-bit Gray code constitutes a minimal-change cycle over all vertices of the n-cube and thus is an optimal solution to the TSP [22]. However, in weighted Hamming spaces, the total cost depends on the number of times each bit is flipped in the binary code. Formally, the problem of finding a minimal-change order C_{opt} of the vertices of the n-cube in a weighted Hamming space can be represented as a minimization problem, where the kth most significant bit has weight w_k and flips $\delta_k(C)$ times over a code C:

$$C_{opt} = \arg\min_{C} \sum_{k=1}^{n} w_k \delta_k(C) \tag{3}$$

The optimal solution C_{opt} to this problem is given by the n-bit binary reflected Gray Code where the bits are arranged in order of weight. The k^{th} most significant bit is flipped a total of $\delta_k(C_{opt}) = 2^{k-1}$ times over the n-bit binary reflected Gray code; thus, assigning the highest weight bits to the most significant positions minimizes the total cost of the ordering.

Algorithm 1: REORDER framework

```
Input  : Crossbar design $\mathcal{D}$, Boolean input vector
         sequence $I_{init}$ of length $L$ with $n$ bits each
Output : Reordered sequence $I_{out}$
$f \leftarrow$VariableFrequencies($\mathcal{D}$);
\\sort in descending order
$\pi_f \leftarrow$ ArgSort($-f$);
for $i = 1$ to $L$ do
    for $j = 1$ to $n$ do
        $I_r[i][j] \leftarrow I_{init}[i][\pi_f[j]]$;
    end
    $X[i] \leftarrow$ BinaryVecToInt($I_r[i]$);
    $G[i] \leftarrow X[i]\ \&\ (X[i] \gg 1)$;
end
\\sort in ascending order
$\pi_I \leftarrow$ ArgSort($G$);
for $i = 1$ to $L$ do
    $I_{out}[i] = I_{init}[\pi_I[i]]$;
end
return $I_{out}$;
```

Motivated by these findings, we compute approximate shortest paths through a subset of vertices of the n-cube by reordering the vertices according to their position in the n-bit binary reflected Gray Code.

Algorithm 1 shows the main steps in the REORDER framework. First, the number of occurrences of each variable is counted. Then, a reordering of the variables is computed such that the variables with the highest frequency in the crossbar design are placed in the most significant bit positions. This reordering is applied to all input vectors in the sequence. Next, the input vectors are converted from binary vectors to the corresponding integer representation

$$x(v) = \sum_{i=0}^{n-1} 2^{n-i} v[i] \tag{4}$$

The position $G_n(x)$ of the integer x in the n-bit Gray code is computed as

$$G_n(x) = x\ \&\ (x \gg 1) \tag{5}$$

where $\&$ denotes the bitwise AND operator and $\gg$ denotes the bitwise right-shift operator. Finally, the input sequence is rearranged in order of ascending Gray codes. Since the initial vector v_0 has a Gray code index of 0, it will always occur first in the resulting sequence. The algorithm can be fully vectorized and computed on a GPU.

V. Image Processing using Flow-Based Computing

Large input sequences occur naturally in applications such as image processing, where a single technique is used to process an entire dataset of images. The order of the processing of the images is not important; all images simply have to be processed. Many image processing techniques can be expressed as a convolution operation on sliding windows of an image; thus, the convolution is executed many times per image. Flow-based computing is particularly suited to this task since the results of the expensive one-time synthesis phase can be re-used to execute the same processing technique on many images.

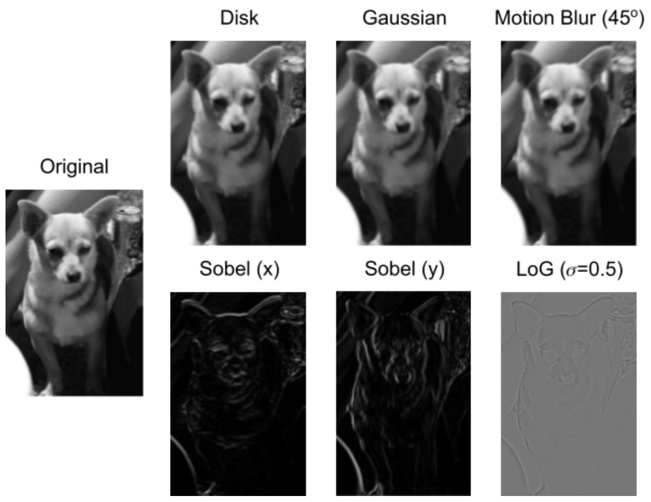

Fig. 4. Result of various image processing techniques applied as discrete 3x3 convolutional filters on ImageNet.

To perform convolution using flow-based computing, it is necessary to synthesize a crossbar design that computes the convolution operation. Given a fixed convolutional filter, we represent the convolution between a sliding window of an image and the filter as a linear combination of pixel values in Verilog code. Then, the Yosys tool [24] is used to convert the high-level Verilog description into a series of Boolean expressions in the form of a Boolean Decision Diagram (BDD) [25]. Finally, we use the COMPACT method [10] to synthesize the crossbar design from the BDD. The resulting crossbar accepts an input-vector containing each pixel value (represented as an 8-bit integer) in the image window corresponding to a non-zero weight in the filter and outputs a single value equal to the result of convolution. The crossbar is then applied sequentially on each window in the image to produce the final result.

Crucially, when applying our REORDER framework across an image dataset, we are not limited to reordering windows within one image; by interleaving sections of different images, we can leverage structural regularities in the dataset to avoid repetitive computation.

A. Image Filtering

Image filtering using discrete kernels can be used to perform a variety of digital processing functions, including noise reduction, motion blur, and edge detection. In image filtering, a small (e.g. 3 × 3), carefully chosen convolutional filter, or kernel, is convolved with an image to produce an output map. Figure 4 shows the effects of using different filters on the resulting image.

We demonstrate image filtering techniques using flow-based computing on the CIFAR-10 dataset. To do so, we synthesize

TABLE I
SYNTHESIS OF CROSSBAR DESIGNS FOR IMAGE FILTERING.

Filter	Rows	Cols	Inputs	Synthesis Time (min)
Disk	658	631	40	2
Gaussian	4354	4366	72	193
LoG ($\sigma = 0.5$)	9985	10114	72	309
Motion Blur (45^o)	2066	2081	56	28
Sobel (x)	3163	2983	48	124
Sobel (y)	2984	3162	48	172

crossbar designs for 6 discrete convolutional filters representing common image processing techniques: Gaussian and disk filters for noise reduction, a motion filter to simulate motion blur, and the Laplacian-of-Gaussian and Sobel filters for edge detection. For each filter, we use Yosys [24] and COMPACT [10] to synthesize a crossbar design to perform convolution with the given weights. We then split each image in the CIFAR-10 dataset into a set of input vectors corresponding to the sliding windows of the image and accumulate the input vectors from all images into a single sequence. Finally, we measure the cost of evaluating the sequence using the synthesized crossbar design before and after reordering. Table I contains the results of synthesis for each crossbar. We measure the dimensions of each crossbar as well as the amount of time required for synthesis.

B. Feature Extraction

Convolution has been used widely in deep learning as a powerful feature extractor for visual inputs. Stacking convolutional layers into a convolutional neural network (CNN) achieves state-of-the-art performance on many image processing tasks with far less model complexity than a traditional fully-connected neural network by reusing the kernel coefficients across sliding windows of the input image. The feature maps produced by CNNs constitute a deep and semantically rich representation of the perceptual content of an image [26]. Furthermore, the latent representations learned by CNNs perform well at transfer learning, i.e. produce meaningful feature maps even on inputs outside the original training distribution. As a result, the output feature map of a convolutional layer which reduces the dimensionality of the input can be seen as a compressed version of the original image.

Previous studies [27] have shown that embedding the first layer of a convolutional network into a CMOS image sensor can leverage this compression property to significantly reduce the bandwidth of communication between sensors and downstream processing units. We extend this work in the beyond-Moore setting by designing a nanoscale crossbar for flow-based computing which can be embedded in-sensor to compress images into feature maps.

Concretely, we achieve this by synthesizing a nanoscale crossbar design to perform convolution with pre-trained weights corresponding to the initial layer of a CNN using Yosys [24] and COMPACT [10]. For layers with multiple output channels, the input must be convolved with one kernel for each output channel; this can be achieved using a single nanoscale crossbar design by synthesizing one crossbar design for each kernel and constructing a composite design in the shape of a block diagonal matrix. The resulting crossbar design can compute the full feature map with a single execution of the evaluation phase of flow-based computing. Table II shows the details of executing the synthesis pipeline for feature extraction. Synthesizing a crossbar design to compute the $13 \times 13 \times 1$ feature map takes much longer than synthesis for other feature maps despite the reduced crossbar size due to the higher magnitude of weights and increase in complexity of the resulting BDD.

TABLE II
SYNTHESIS OF CROSSBAR DESIGNS FOR FEATURE EXTRACTION.

Feature Map Size	Rows	Cols	Inputs	Synthesis Time (min)
$13 \times 13 \times 1$	3750	3527	45	435
$13 \times 13 \times 2$	7266	7268	45	102
$13 \times 13 \times 3$	8668	8542	45	115

TABLE III
FEATURE EXTRACTOR QUALITY

Feature Map Size	Compression Ratio	Accuracy
$13 \times 13 \times 1$	86.53%	98.00%
$13 \times 13 \times 2$	73.05%	98.12%
$13 \times 13 \times 3$	59.58%	98.28%
$28 \times 28 \times 1$	0.00%	98.45%

Additionally, due to crossbar constraints, we must quantize the inputs, network weights, and outputs to a fixed precision chosen to minimize crossbar synthesis time and data bandwidth without significantly damaging the quality of the resulting feature map. In our experiments, all inputs, weights, and outputs are quantized to 5 bits per pixel. We observe that, in all tested cases, the loss in downstream classification accuracy due to feature map quantization does not exceed 0.3%.

We measure the effectiveness of our flow-based feature extractor on MNIST for feature maps of varying sizes. Table III shows the accuracy achieved by a downstream classifier CNN as a function of the compression ratio, measured as the reduction in the size of the feature map (in bytes) relative to the original image. Our method is able to compress images by up to 86.53% with minimal impact on the performance of the downstream classifier.

VI. EXPERIMENTAL RESULTS

Section VI-A details the architecture used in our REORDER framework. In Section VI-B, we evaluate and compare different TSP heuristics. In Section VI-C, we evaluate the effectiveness of the the REORDER framework. The experimental evaluation was performed on a machine with an Intel® Core™ i9-9900K CPU @ 3.60GHz CPU with 8 cores and a NVIDIA TITAN RTX GPU.

A. Architecture

We present the overview of the architecture of the REORDER framework in Figure 6.

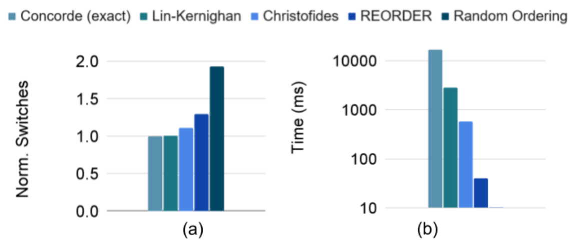

Fig. 5. Comparison of the optimality (a) and latency (b) of approximate TSP solving algorithms for problems of size N=1000.

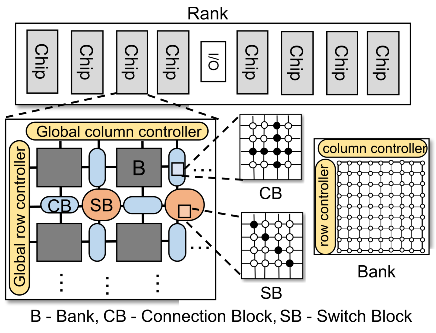

Fig. 6. Architecture of the REORDER framework.

Each rank of the architecture consists of several chips and an I/O interface as shown on the top of the Figure 6. Each chip contains several crossbar banks connected in an island-style architecture. The routing architecture is shown at the bottom-left of the figure. The connection blocks (CB) routs signals between the neighbouring blocks. The switch blocks (SB) supports cross-architecture communication. The cross-sections of the crossbar bank, the CB and the SB is shown in bottom-right of the figure. Global and local controllers are used to program the crossbar banks and routing blocks. The area-power cost of different architectural components are summarized in Table IV. The parameter costs are appropriately adapted from previous works [28], [29], [30]. Whenever the computational space of a task exceeds the crossbar bank dimension, the computation is partitioned and assigned to multiple banks.

TABLE IV
AREA-POWER COST OF ARCHITECTURAL COMPONENTS

Component	Parameter	Specs	Area	Power
Crossbar Bank	size	1024×1024	1600 μm^2	19.2 mW
Controller	# unit	1	411 μm^2	0.67 mW
Sense Amp.	# unit	1024	57.12 μm^2	2.32 mW
Connection Block	bandwidth	1 kb	4.1 μm^2	0.05 mW
Switch Block	bandwidth	1 kb	16.35 μm^2	0.21 mW
Bus	bandwidth	1 kb	125.6 mm^2	104 mW

B. Evaluation of TSP Algorithms

First, we evaluate our proposed heuristic against several approximate methods for computing shortest Hamiltonian paths. Our implementation is parallelized to run on a GPU using PyTorch and CUDA. We evaluate all methods on a sequence of input vectors of size $N = 1000$ and measure the time required to generate the approximate shortest path. The results are shown in Figure 5. Our method can find inexpensive paths (a) with negligible computational overhead (b) compared to pre-existing algorithms. We are able to achieve significant improvements over general-purpose TSP solving methods by taking advantage of the additional properties of binary weighted Hamming spaces to efficiently find shortest paths.

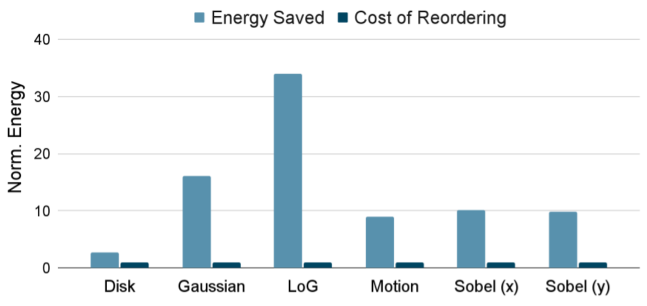

Fig. 7. Comparison of the cost of input vector reordering with the energy saved by executing the reordered sequence.

Figure 7 shows the cost of computing the approximate TSP solution using our proposed Gray heuristic relative to the energy saved by executing the reordered input vector sequence. Our method can find approximate TSP solutions within milliseconds, whereas general-purpose TSP algorithms take several seconds and produce only slightly shorter paths. Furthermore, other methods scale poorly: even heuristic TSP algorithms require polynomial time in the number of input vectors to find an approximate shortest path, whereas our method only takes log-linear time. The efficiency of our heuristic results in a small energy cost relative to the energy saved by reordering.

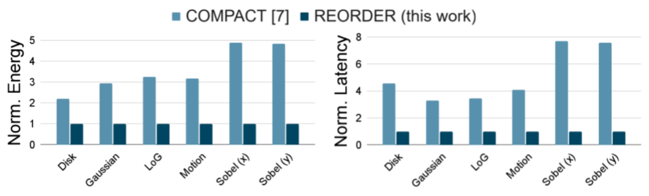

Fig. 8. Estimated energy cost (a) and latency (b) of image filtering using the REORDER framework.

C. REORDER Framework

We measure the performance of our framework as the total energy cost of computing the ordering and subsequently evaluating the reordered input vector sequence. Because memristor switching accounts for most of the time and energy consumption required to perform flow-based computing, we estimate the latency and energy efficiency of crossbar evaluations by assigning a fixed time and energy cost of 50.88 ns and 3.91 nJ respectively per WRITE operation [31].

Figure 8 shows the estimated energy cost (a) and latency (b) of performing image filtering on the CIFAR-10 dataset using the REORDER framework relative to naïvely evaluating the input vector sequence (i.e. in an arbitrary order) using flow-based computing. Our framework requires 78% fewer switches

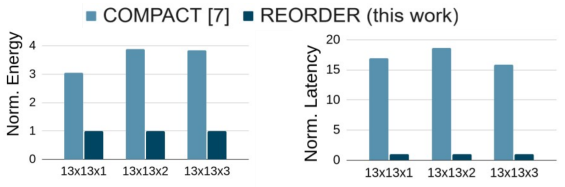

Fig. 9. Estimated energy cost (a) and latency (b) of feature extraction using the REORDER framework.

to process the dataset, decreasing latency and energy usage by 78% and 69%, respectively. In general, we observe greater energy savings for larger crossbars due to the higher number of switches prevented relative to the length of the sequence. On the other hand, we find that the time required to pre-compute the reordering is negligible compared to the cost of evaluation for all designs tested.

We repeat the energy and latency analysis for our REORDER framework for feature map extraction on the MNIST dataset and display the results in Figure 9. On average, the REORDER framework achieves a 94% decrease in latency and a 72% decrease in energy.

VII. Conclusion

In this paper, we present the REORDER framework for reducing power dissipation in flow-based computing systems via input vector sequence reordering and demonstrate how image processing techniques can be executed efficiently using our framework. We show that the additional cost of sequence reordering is negligible compared to the time and energy saved by executing the reordered sequence on a crossbar array. We achieve speedup and energy savings of 78% and 69% respectively for image filtering and savings of 94% and 72% respectively for feature map extraction. Further directions of research include implementation of the reordering algorithm on energy-efficient hardware such as FPGA or ARM processors.

References

[1] R. S. Williams, "What's next?[the end of moore's law]," *Computing in Science & Engineering*, vol. 19, no. 2, pp. 7–13, 2017.

[2] J. M. Shalf and R. Leland, "Computing beyond moore's law," *Computer*, vol. 48, no. 12, pp. 14–23, 2015.

[3] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[4] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[5] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[6] S. Pi, C. Li, H. Jiang, W. Xia, H. Xin, J. J. Yang, and Q. Xia, "Memristor crossbar arrays with 6-nm half-pitch and 2-nm critical dimension," *Nature nanotechnology*, vol. 14, no. 1, pp. 35–39, 2019.

[7] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magic—memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[8] 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," *IEEE Transactions on VLSI Systems*, vol. 22, no. 10, pp. 2054–2066, 2013.

[9] B. Feinberg, U. K. R. Vengalam, N. Whitehair, S. Wang, and E. Ipek, "Enabling scientific computing on memristive accelerators," in *2018 ACM/IEEE 45th ISCA*, pp. 367–382, IEEE, 2018.

[10] S. Thijssen, S. K. Jha, and R. Ewetz, "Compact: Flow-based computing on nanoscale crossbars with minimal semiperimeter," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 232–237, IEEE, 2021.

[11] D. Chakraborty and S. K. Jha, "Design of compact memristive in-memory computing systems using model counting," in *2017 IEEE ISCAS*, pp. 1–4, IEEE, 2017.

[12] A. Velasquez and S. K. Jha, "Automated synthesis of crossbars for nanoscale computing using formal methods," in *Proceedings of the 2015 IEEE/ACM NANOARCH´ 15*, pp. 130–136, IEEE, 2015.

[13] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital cmos circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498–523, 1995.

[14] M. A. Zidan, H. Omran, A. Sultan, H. A. Fahmy, and K. N. Salama, "Compensated readout for high-density mos-gated memristor crossbar array," *IEEE Transactions on Nanotechnology*, vol. 14, no. 1, pp. 3–6, 2014.

[15] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-path constraints in memristor crossbar arrays," in *2013 IEEE International Symposium on Information Theory*, pp. 156–160, IEEE, 2013.

[16] Z. Alamgir, K. Beckmann, N. Cady, A. Velasquez, and S. K. Jha, "Flow-based computing on nanoscale crossbars: Design and implementation of full adders," in *2016 IEEE ISCAS*, pp. 1870–1873, IEEE, 2016.

[17] S. Chakravarty and V. Dabholkar, *Minimizing power dissipation in scan circuits during test application*. Department of Computer Science, State University of New York at Buffalo, 1994.

[18] P. Flores, J. Costa, H. Neto, J. Monteiro, and J. Marques-Silva, "Assignment and reordering of incompletely specified pattern sequences targetting minimum power dissipation," in *Proceedings Twelfth International Conference on VLSI Design.(Cat. No. PR00013)*, pp. 37–41, IEEE, 1999.

[19] D. L. Applegate, R. E. Bixby, V. Chvátal, W. Cook, D. G. Espinoza, M. Goycoolea, and K. Helsgaun, "Certification of an optimal tsp tour through 85,900 cities," *Operations Research Letters*, vol. 37, no. 1, pp. 11–15, 2009.

[20] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations research*, vol. 21, no. 2, pp. 498–516, 1973.

[21] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[22] G. Cohen, S. Litsyn, and G. Zemor, "On the traveling salesman problem in binary hamming spaces," *IEEE Transactions on Information Theory*, vol. 42, no. 4, pp. 1274–1276, 1996.

[23] S. Arora, "Polynomial time approximation schemes for euclidean tsp and other geometric problems," in *Proceedings of 37th Conference on Foundations of Computer Science*, pp. 2–11, IEEE, 1996.

[24] C. Wolf, "Yosys open synthesis suite," 2016.

[25] S. B. Akers, "Binary decision diagrams," *IEEE Transactions on computers*, vol. 27, no. 06, pp. 509–516, 1978.

[26] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595, 2018.

[27] G. Datta, S. Kundu, Z. Yin, R. T. Lakkireddy, P. A. Beerel, A. Jacob, and A. R. Jaiswal, "P2m: A processing-in-pixel-in-memory paradigm for resource-constrained tinyml applications," *arXiv preprint arXiv:2203.04737*, 2022.

[28] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *2019 ACM/IEEE 46th ISCA*, pp. 802–815, IEEE, 2019.

[29] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "Vteam: A general model for voltage-controlled memristors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 8, pp. 786–790, 2015.

[30] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[31] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE International Symposium on*

High Performance Computer Architecture (HPCA), pp. 541–552, IEEE, 2017.