COMPACT: Flow-Based Computing on Nanoscale Crossbars with Minimal Semiperimeter and Maximum Dimension

Sven Thijssen[1], Sumit Kumar Jha[2], and Rickard Ewetz[3]
[1,2]*Department of Computer Science,* [3]*Department of Electrical and Computer Engineering*
[1,3]*University of Central Florida, Orlando, USA,* [2]*University of Texas at San Antonio, San Antonio, USA*
sven.thijssen@knights.ucf.edu, sumit.jha@utsa.edu, rickard.ewetz@ucf.edu

***Abstract*—In-memory computing is a promising solution strategy for data-intensive applications to circumvent the *von Neumann bottleneck*. Flow-based computing is the concept of performing in-memory computing using sneak paths in nanoscale crossbar arrays. The limitation of previous work is that the resulting crossbar representations have large size. In this paper, we present a framework called COMPACT for mapping Boolean functions to crossbar representations with minimal semiperimeter (the number of wordlines plus bitlines) and/or maximum dimension (the maximum of the wordlines or bitlines). The COMPACT framework is based on an analogy between binary decision diagrams (BDDs) and nanoscale memristor crossbar arrays. More specifically, nodes and edges in a BDD correspond to wordlines/bitlines and memristors in a crossbar array, respectively. The relation enables a Boolean function represented by a BDD with n nodes and an odd cycle transversal of size k to be mapped to a crossbar with a semiperimeter of n+k. The k extra wordlines/bitlines are introduced due to crossbar connection constraints, i.e. wordlines (bitlines) cannot directly be connected to wordlines (bitlines). Moreover, there exists a trade-off between the semiperimeter and maximum dimension. Consequently, COMPACT can sometimes reduce the maximum dimension by slightly increasing the length of the semiperimeter. We also extend COMPACT to handle multi-output functions using shared binary decision diagrams (SBDDs) and alignment constraints on the inputs and outputs. Compared with the state-of-the-art mapping technique, the semiperimeter and maximum dimension are reduced with 55% and 85%, respectively. The area, power consumption, and computation delay are reduced with 89%, 19%, 56%, respectively.**

***Index Terms*—flow-based, in-memory, computing, memristor, crossbar, synthesis**

I. Introduction

Many modern computer architectures are based on the concepts defined in *First draft of a report on the EDVAC* by von Neumann [1]. These computer architectures suffer from the *von Neumann bottleneck*. This bottleneck is an inevitable consequence of the data transfer between separated memory units and processing units [2]. The in-memory computing paradigm aims to solve this bottleneck by unifying memory storage and computation.

In 1971, L. Chua introduced a new circuit element, which he called *memristor* [3]. In 2008, Hewlett Packard Laboratories was the first to finally develop a physical model of this fourth fundamental circuit element [4]. This led to the development of new computing paradigms using memristors, such as *material-based implication logic* (IMPLY) [5], *memory-aided logic (MAGIC)* [6] and *flow-based computing* [7]. Each of these approaches have their respective strengths and weaknesses. For IMPLY-based logic, a major drawback is the number of complex computational steps required to synthesize a Boolean function [8], [9]. More specifically, parallelism is inherently limited for IMPLY-based logic, resulting in long, sequential executions of a Boolean function. IMPLY logic is a non-stateful logic, which entails that intermediate evaluations depend upon previous intermediate evaluations, resulting in an almost sequential evaluation. On the other hand, the parallelism within the MAGIC-style is fundamentally limited [10]. MAGIC relies on NOR operations to evaluate a Boolean function. Also here, the stateful logic requires consecutive intermediate evaluations. This dependency also results in long sequential evaluations.

The flow-based computing paradigm is based on taking advantage of the natural flow of electrical current. By programming the resistance of memristors in a crossbar based on Boolean variables, Boolean functions can be evaluated by applying a high potential to the bottom most wordline and measuring the output current from a predefined wordline. The function evaluates to true if and only if there exists at least one path from the input to the output containing only memristors in the low resistive state.

Flow-based computing has been explored based on negation normal form (NNF) [7], disjunctive normal form (DNF), conjunctive normal form (CNF) [11], simulated annealing [12] and satisfiability modulo theories (SMT) [13]. Unfortunately, these initial methods were computationally expensive or resulted in crossbar representations with large size. More specifically, the work in [14] is only capable of synthesizing 1-bit adders, and the work in [15] results in large crossbar designs. To overcome these shortcomings, recent studies are based on mapping binary decision diagrams (BDDs) to crossbars using inductive staircase structures. The mapping of BDDs in the form of reduced ordered binary decision diagrams (ROBDD) and free binary decision diagrams (FBDD) has been explored [16], [17], [18], [19]. The staircase structures span from the bottom-left corner to the top-right corner of the crossbar. These inductive techniques are promising because

This work was in part supported by NSF awards CCF-1755825, CNS-1908471, and CCF-1822976.

both the number of rows and columns can be proved to grow linearly with the number of nodes in the BDD [16]. In particular, the dimensions of the nanoscale crossbar is upper bounded by $3n$ by n [16], where n is the number of nodes in the BDD. Fortunately, it can be observed that the crossbar representations have a dimension of closer to n by n in practice. Nevertheless, some rather simple Boolean functions still result in crossbars with excessive size. The size of a crossbar representation is measured in terms of the semiperimeter (the number of wordlines plus bitlines) and maximum dimension (the maximum of the number of wordlines and bitlines). In [18] C code is compiled into BDDs using the LLVM compiler and crossbars designs are subsequently constructed using the automated synthesis method in [16].

In this paper, we propose a framework called COMPACT for mapping BDDs into minimal crossbar representations. COMPACT minimizes the weighted sum of the crossbar semiperimeter and maximum dimension. The framework is based on an analogy between BDDs and nanoscale memristor crossbars. More specifically, nodes and edges in a BDD correspond to wordlines/bitlines and memristors in a crossbar, respectively. The relation enables a BDD with n nodes and an odd cycle transversal of size k to be mapped into a crossbar representation with a semiperimeter of $n+k$. There exists a trade-off between the semiperimeter and the maximum dimension. Consequently, the maximum crossbar dimension can sometimes be reduced by increasing the length of the semiperimeter.

COMPACT determines minimal crossbar representations by viewing the BDD as a graph and formulating a node labeling problem, called the *VH-labeling problem*. The node labeling problem defines if a node in the BDD is mapped to a wordline, a bitline, or both a wordline and a bitline. In this paper, we introduce two methods for solving the node labeling problem. In the first method, the semiperimeter is minimized by casting the problem to an odd cycle transversal problem that can be solved using a minimum vertex cover. In the second method, we build upon the theoretical understanding we have gained from the first approach to minimize the weighted sum of the semiperimeter and the maximum dimension, which is performed using a MIP formulation.

For multi-input multi-output functions, COMPACT can be directly applied to shared binary decision diagrams (SBDDs) instead of multiple separate single-output BDDs, which further improves the crossbar size. COMPACT is also capable of aligning the function outputs on the crossbar representation. Compared with previous works on flow-based computing, the experimental results show that the semiperimeter, maximum dimension, and area is reduced by 55%, 85% and 89% on average, respectively. Furthermore, power consumption and computation delay are reduced by 19% and 56% on the average. Compared with CONTRA, the state-of-the-art framework for MAGIC-based in-memory computing, COMPACT reduces power consumption and computation delay by 55% and 87% on the average for the EPFL control benchmarks.

The remainder of the paper is organized, as follows: background in Section II and problem formulation in Section III. The BDD-crossbar analogy is given in Section IV. The COMPACT framework for single-output functions is introduced in Section V. In Section VI, the two methods for solving the VH-labeling problem are proposed. The COMPACT framework is extended to multi-output functions in Section VII. The paper is concluded with experimental results in Section VIII.

II. Background

A. Binary decision diagrams

A BDD is a graph representation of a Boolean function. The internal nodes are Boolean variables and the leaf nodes are either `0' or `1'. A BDD is evaluated by traversing the nodes along a path from the root node to a leaf node. At each internal node of the BDD, depending on the value of the Boolean variable, one must decide which path to follow to evaluate an instance [20]. ROBDDs and FBDDs are extensions of BDDs for multi-input single-output functions that are optimized to minimize number of nodes and edges. BDDs can be extended to multi-input multi-output functions using shared binary decision diagrams (SBDDs). In SBDDs, multiple ROBDDs are merged together [21].

B. Nanoscale memristor crossbars

A memristor crossbar is a two-dimensional array consisting of two layers of nanowires. The horizontal nanowires are wordlines and the vertical nanowires are bitlines. Each layer is a set of parallel nanowires with each layer being perpendicular to one another. A memristor connects one layer with another at the intersections of the perpendicular nanowires [7]. Memristors with high endurance and fast switching speed have been demonstrated for memory applications [22]. A major concern for memory applications is the occurrence of currents on sneak paths, which reduces the effective write voltage [22]. In contrast, flow-based computing is based on leveraging the sneak paths to perform computation.

A nanoscale crossbar can have a matrix representation or a graph representation. More precisely, a nanoscale crossbar of N rows and M columns can be represented by a complete bipartite graph $K_{M,N}$. In Figure 1(a), a matrix representation is given for a nanoscale crossbar of dimension 4 by 4, and in Figure 1(b) its corresponding bipartite graph is given.

(a) Matrix representation (b) Bipartite graph

Fig. 1. Representations of nanoscale crossbars

C. Flow-based in-memory computing

Flow-based computing is based on evaluating Boolean functions using the sneak currents that naturally occur in nanoscale

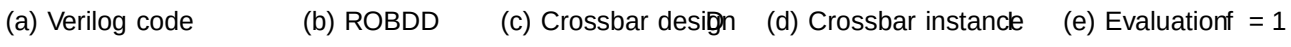


Fig. 2. Overview of the flow-based computing paradigm

crossbars. Computing within the paradigm is performed using a one-time costly initialization phase and an efficient and fast evaluation phase, which is illustrated in Figure 2.

In the initialization phase, a Boolean function is converted into a crossbar representation. The Boolean function is specified using a Verilog, BLIF or PLA file. A Boolean function $f = (a \wedge b) \vee c$ is shown in Figure 2(a). Next, a BDD representation of is constructed using ABC/CUDD [23], which is shown in Figure 2(b). Previous work mainly utilized BDDs in the form of ROBDDs [16]. The next step is to map the BDD into a crossbar representation. This involves assigning each memristor in the crossbar to logical `0' or `1' or a Boolean variable from $\{a, b, c\}$ or the negation of a Boolean variable from $\{\neg a, \neg b, \neg c\}$. An input port and an output port are also assigned to the crossbar. A crossbar representation that realizes the BDD in Figure 2(b) is shown in Figure 2(c).

In the evaluation phase, the Boolean function is evaluated using the crossbar representation and an instance of the Boolean variables. The first step is to program the memristors in the crossbar based on the instance of the Boolean variables. Memristors in the crossbar are programmed to have low (high) resistance if the assigned logic expression is true (false). In the example, the crossbar instantiation for $a = 1$, $b = 1$ and $c = 0$ is shown in Figure 2(d). Next, an input voltage V_{in} is applied to the bottom most wordline and is evaluated by measuring the output voltage V_{out} across a sensing resistor, which is connected to ground. In the example, it can be observed that there exists a path from the input to the output that only contains memristors with low resistance. Therefore, the output voltage is high and the Boolean function evaluates to true, which is shown in Figure 2(e).

III. Problem formulation

The COMPACT framework aims to design a valid crossbar representation for a Boolean function . A crossbar representation is a valid representation of a Boolean function if and only if for every instance of the Boolean variables, there exists a path from the input to the output using only memristors in the low resistive state where evaluates to true. We propose to find valid crossbar designs that minimize the following objective:

$$\alpha S + (1 - \alpha) D, \tag{1}$$

where is a user-defined parameter within $[0, 1]$. and are semiperimeter and maximum dimension of the crossbar design, respectively. The parameter is to balance the two terms in the objective. The maximum dimension is defined as $\max(R, C)$ where R is the number of rows (wordlines), and C is the number of columns (bitlines). Similarly, is defined to be equal to $R + C$. There are two special cases for the parameter . If is equal to one, the semiperimeter is minimized without regard to the maximum dimension. When is equal to zero, then the maximum dimension is minimized without regard to the semiperimeter.

The motivation for including the semiperimeter in the objective is to synthesize crossbars as small as possible. The motivation for including the maximum dimension is that manufactured crossbars tend to be square [24].

We further introduce constraints for the alignment of the inputs and outputs. In flow-based computing, the inputs and the outputs are assigned to wordlines. More specifically, the input is assigned to the bottom-most nanowire and the output(s) are assignment to the top-most nanowire(s).

Note that it is trivial to modify our problem formulation and COMPACT to handle specified constraints on the rows and columns. For such problem formulations, COMPACT would generate a valid design or return that the specified design constraints are infeasible.

IV. Analogy between BDDs and crossbars

The COMPACT framework in this paper is based on the observation that an analogy exists between BDDs and memristor crossbars. More specifically, the nodes and edges within a BDD correspond to the bitlines/wordlines and memristors in a crossbar, respectively. Theoretically, a BDD with nodes can be mapped to a crossbar with a semiperimeter of . However, a memristor crossbar places inherent constraints on the connections realized by the memristors; wordlines cannot be connected directly to wordlines and bitlines cannot be connected directly to bitlines. Therefore, extra hardware resources (intermediate bitlines or wordlines) are needed to realize such connections. One way to circumvent the connection constraint problem is to map each node to both a wordline and a bitline. However, this leads to a crossbar representation with a semiperimeter of . The COMPACT framework aims

to find smaller crossbar designs by mapping as few nodes as possible to both wordlines and bitlines while resolving the connection constraints. In fact, COMPACT is capable of assigning the fewest possible BDD nodes to both wordlines and bitlines, which results in crossbar representations with minimal semiperimeter. On the other hand, the crossbar design with minimal semiperimeter may be highly unbalanced. We observe that it may be necessary to resolve the connection constraints using additional hardware (bitlines/wordlines) to find more square-like crossbar designs.

V. The COMPACT framework

The flow of the COMPACT framework is shown in Figure 3 and illustrated with an example in Figure 4. The input to the framework is a Boolean multi-input single-output function represented using a ROBDD, which is illustrated in Figure 4(a). The output of the framework is a crossbar representation of the Boolean function. COMPACT is extended to multi-input multi-output functions with alignment constraints in Section VII.

The main steps of COMPACT are graph pre-processing, VH-labeling and crossbar mapping. In the graph pre-processing step, the BDD is converted into a graph representation. In the VH-labeling step, each node in the graph is assigned a label V, H or VH, indicating if they will be mapped to a vertical bitline (V), horizontal wordline (H), or both a vertical bitline and a horizontal wordline (VH). The labels VH are introduced to handle the connection constraints imposed by the nanoscale crossbar. In the crossbar mapping step, nodes in the graph are bound to specific wordlines/bitlines according to the assigned labels. The edges in the graph are correspondingly assigned to memristors in the crossbar.

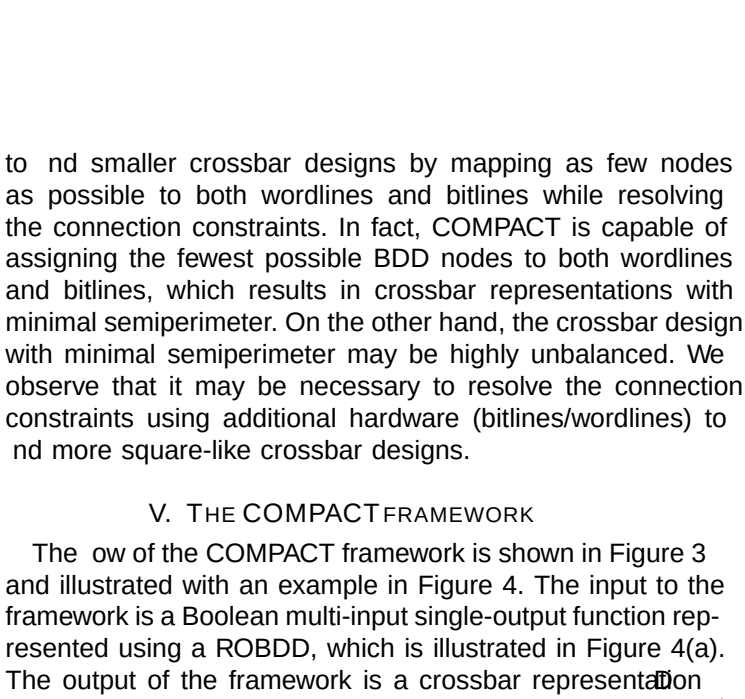


Fig. 3. Overview of the COMPACT framework

A. Graph pre-processing

In this section, the input BDD is converted into an undirected graph G. This is performed by first removing terminal node `0' and its incoming edges. The zero can be removed because flow-based computing aims to only capture the `1' output. Finally, the graph representation is obtained by mapping each node/edge in the BDD to an node/edge in an undirected graph. The resulting graph of the BDD in Figure 4(a) is shown in Figure 4(b).

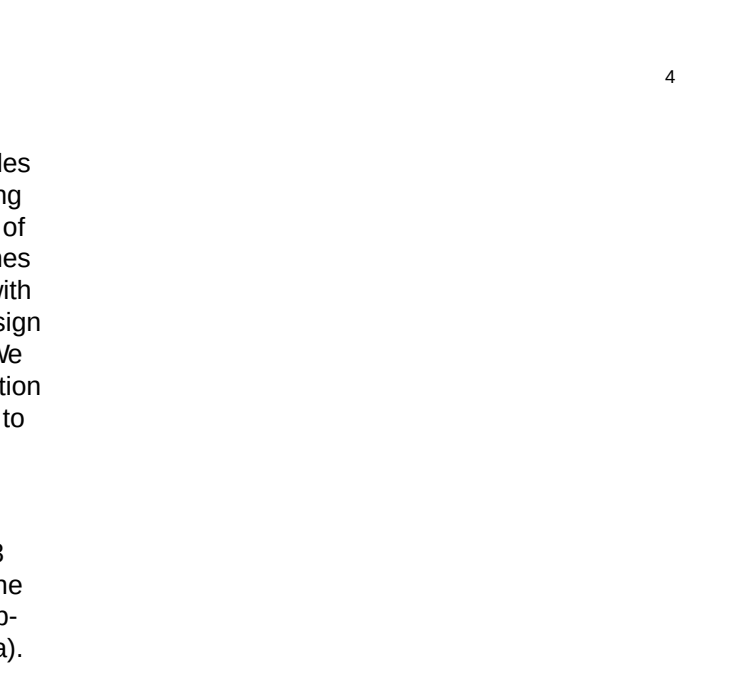


Fig. 4. Example of the COMPACT framework

B. VH-labeling

The input to the VH-labeling step is the undirected graph G. The step involves assigning a label V, H, or VH to each node in the graph. The labeled graph is illustrated in Figure 4(c). The labels are introduced to ensure that all edges in the graph can later be realized using a memristor in the subsequent crossbar mapping step, i.e., preemptively handling the connection constraints. The labeling solution directly defines both the semiperimeter S and the maximum dimension D. In this section, we define the VH-labeling problem as an mathematical optimization problem. Next, we provide two solutions to solving the VH-labeling problem in Section VI.

The VH-labeling problem: Let $G = (U, E)$ be the undirected graph, where U is a set of vertices and E is a set of edges E, serving as input to the VH-labeling step. The VH-labeling problem consists of assigning a label $\{V, H, VH\}$ to each node in the graph such that the connection constraints are satisfied. First, we introduce an objective that minimizes the semiperimeter S by minimizing the number of VH labels. Next, we provide a solution to minimize the weighted sum of the semiperimeter S and the maximum dimension D.

We formally define the VH-labeling problem for minimizing the semiperimeter S as follows:

$$\begin{aligned} \min \quad & |\{v \mid v = L^{-1}(VH)\}| \\ \text{s.t.} \quad & : (L(u) = V \wedge L(v) = V); \qquad (u,v) \in E \\ & : (L(u) = H \wedge L(v) = H); \qquad (u,v) \in E \end{aligned} \tag{2}$$

where u and v are vertices in U. $L : U \to \{V, H, VH\}$ is the label given to node v.

The objective directly minimizes the number of VH labels, which explicitly defines the semiperimeter S of the resulting crossbar representation. The semiperimeter is equal to $n + k$ if the graph has n nodes and k VH labels. The area is implicitly optimized by minimizing the semiperimeter. The two constraints ensure that no adjacent nodes in the graph G are assigned (V, V) or (H, H) labels, as it would be impossible to connect the corresponding bitlines or wordlines using a memristor.

For the second method, the weighted objective of both the semiperimeter S and the maximum dimension D is minimized. The weighted objective is rewritten across multiple lines for clarity. The constraints are the same as in Eq 2.

$$\begin{aligned} \min \quad & S + (1-\ )D \\ & S = |\{v \mid v = L^{-1}(V \text{ or } H)\}|; \\ & D = \max(R, C); \\ & R = |\{v \mid v = L^{-1}(H) \text{ or } v = L^{-1}(V \text{ or } H)\}|; \\ & C = |\{v \mid v = L^{-1}(V) \text{ or } v = L^{-1}(V \text{ or } H)\}|; \end{aligned} \tag{3}$$

C. Crossbar mapping

In the crossbar mapping step, we bind the graph G to a crossbar representation D according to the assigned labels, which ensures that the connection constraints can be satisfied.

The mapping is performed by using a node assignment step and an edge assignment step. In the node assignment step, each node in the graph is assigned to a bitline, wordline, or both a bitline and a wordline according to the label in the graph, i.e., nodes labeled V (H) are assigned to bitlines (wordlines). Nodes labeled V or H are assigned to both a bitline and a wordline. However, these wordlines and bitlines are supposed to be connected. Therefore, we also program the memristor in the intersection of the corresponding wordline and bitline to have low resistance or `1'. The crossbar representation following the node assignment step is shown in Figure 4(d).

In the edge assignment step, each edge in the graph is mapped to a memristor in the crossbar such that it connects the bitlines and wordlines that correspond to the nodes in the graph. Following the node assignment step, the edge assignment step maps the variables and their negations onto the crossbar representation, as shown in Figure 4(e). The output is a crossbar design D for a Boolean function using the COMPACT framework.

VI. Solving the VH-labeling problem

In Section VI-A, we solve the VH-labeling problem while minimizing the semiperimeter based on Eq 2. In Section VI-B, we solve the VH-labeling problem while minimizing the weighted objective based on Eq 3. The first solution method provides an intuitive approach to solve the VH-problem using graph theory. The second approach leverages the theoretical insights of the first approach and solves the problem using a MIP formulation.

A. Minimal semiperimeter

In this section, we provide an optimal algorithm to solve the VH-labeling problem, which results in crossbar representations with the minimal semiperimeter. Next, we provide the proof of correctness.

Fig. 5. Example of the VH-labeling for a minimal semiperimeter

1) The algorithm: If G is bipartite, it is trivial to determine an optimal solution to Eq 2 using 2-coloring. The colors would be the labels V and H. If G is not bipartite, no 2-coloring exists [25]. Hence, not every pair of adjacent nodes can be given a label V and H. Consequently, a V/H label must be assigned to at least one node. A necessary condition for a graph to be bipartite is that it does not contain an odd-length cycle [25].

Our optimal solution to the VH-labeling problem lies in the observation that solving Eq 2 is equivalent to finding the largest induced bipartite subgraph G_B of the graph G. The nodes in G that are not part of G_B are the nodes labeled V/H. The nodes in G_B can trivially be labeled V and H using 2-coloring. Moreover, finding the largest induced bipartite subgraph is equivalent to the odd-cycle transversal problem.

Definition 1 (Odd Cycle Transversal)**.** The odd cycle transversal (OCT) of an undirected graph $G = (V, E)$ is a set $X \subseteq V$, $|X| \geq k$, such that $V \setminus X$ is a bipartite graph [26].

We use lemma 1 to find such odd cycle transversal for G.

Lemma 1. A graph $G = (V, E)$ with $|V| = n$ has an odd cycle transversal X, $|X| \leq k$, if and only if $P = G \square K_2$ has a vertex cover $V_C(P)$ such that $|V_C(P)| \leq n + k$ [26].

We leverage this solution method to finding a minimum vertex cover of P and thus to finding a smallest odd cycle transversal of G. Graph G. In Figure 5(b), we show the graph $P = G \square K_2$, i.e. the Cartesian product of G in Figure 5(a) with K_2. K_2 is a graph with two nodes connected by an edge. The resulting graph P contains two duplicates of graph G. Above this, a node's two duplicates are connected by an edge. If the nodes in K_2 are given a name 0 and 1, then the name of a node in P is the concatenation of the node's respective name in G and either 0 or 1. For example, node a in graph G is duplicated in two nodes $a0$ and $a1$ in graph P. A vertex cover $V = V_C(G)$ for a graph $G = (U, E)$ is a set of nodes $V \subseteq U$ such that for each edge $e = (u, v) \in E$ at least one node u or

v is in W. The minimum vertex cover problem can be solved using integer linear programming (ILP) [27]. The minimum vertex cover of P in Figure 5(b) is shown in Figure 5(c). If both products v_0 and v_1 of a node v are present in the vertex cover W, then v belongs to the odd cycle transversal X of G. It can be observed that both $b0$ and $b1$ belong to the vertex cover in Figure 5(d), which results in that the node is part of the OCT and is labeled VH in Figure 5(e). Finally, the largest induced bipartite subgraph G_B is obtained by only considering the nodes in G which are not labeled VH. The labeling of G_B is performed using traditional 2-coloring, as shown in Figure 5(f).

2) The proof: The COMPACT framework is based on the analogy between a nanoscale crossbar and a BDD. Given that a crossbar is a bipartite graph where the nanowires correspond to the nodes in the undirected graph G of the BDD, and given that the memristors correspond to the edges in the undirected graph G of the BDD, the framework finds the largest induced bipartite subgraph G_B of G. The nodes in G_B are labeled using a 2-coloring. Each color corresponds to a label V or H, denoting whether the node will be assigned to a vertical nanowire or a horizontal nanowire, respectively. Nodes outside G_B belong to an odd cycle transversal of the undirected graph G.

The nodes belonging to the OCT X are given a label VH and are consequently mapped to both a horizontal and a vertical nanowire. Since the ILP formulation in Eq 2 results in the smallest odd cycle transversal X using a minimum vertex cover, then by definition the number of nodes labeled VH is minimal. This completes the proof.

B. Weighted objective

In the following sections, we discuss solving the VH-labeling problem with the weighted objective. In the Section VI-B1, we introduce a MIP formulation for solving Eq 3. In Section VI-B2, we analyze why the MIP formulation is capable of finding crossbar designs with smaller maximum dimension.

1) The MIP formulation: In this section, we provide a MIP formulation to solve the VH-labeling problem in Eq 3. For each node $i \in U$, we introduce variables x_i^V and x_i^H. When $x_i^V = 1$, then the node will be mapped vertically (bitline). When $x_i^H = 1$, then the node will be mapped horizontally (wordline). When both $x_i^V = 1$ and $x_i^H = 1$, then the node will be mapped both vertically and horizontally. Next, the MIP problem is formulated, as follows:

$$
\begin{aligned}
\min \quad & \alpha S + (1-\alpha) D \\
\text{s.t.} \quad & S = \sum_{i \in U} x_i^V + x_i^H \\
& R = \sum_{i \in U} x_i^V ; \\
& C = \sum_{i \in U} x_i^H ; \\
& D \geq R; \\
& D \geq C; \\
& x_i^V + x_j^H \geq 2 - 2x_{ij} ; \qquad \forall (i,j) \in E \\
& x_i^H + x_j^V \geq 2 - 2(1 - x_{ij}); \qquad \forall (i,j) \in E
\end{aligned}
\tag{4}
$$

where $x_i^V, x_i^H \in \{0,1\}, \forall i \in U$, $\alpha \in [0,1]$, $S \in \mathbb{N}$, $D \in \mathbb{N}$, and U is the set of nodes for the undirected graph G. $x_{ij} \in \{0,1\}$ is a binary helper variable for each edge $(i,j) \in E$. The objective in Eq 4 directly minimizes the objective in Eq 2 and Eq 3. The first five constraints in Eq 4 are used to define S and D based on the variables x_i^V and x_i^H. The last two constraints are used to handle the the constraints in Eq 2 (connection constraints). Now we elaborate further on these two constraints.

The main idea is that the connection between node i and node j for an edge (i,j) must be either an V-H or an H-V connection. The helper variable x_{ij} specifies if the connection is of type V-H or H-V. Basically, x_{ij} will select to activate one of the following two constraints:

$$x_i^V + x_j^H \geq 2 \tag{5}$$

$$x_i^H + x_j^V \geq 2 \tag{6}$$

2) Analysis of MIP formulation: In Section VI-A, we have introduced an algorithm for constructing crossbar designs with minimal semiperimeter. However, the resulting crossbar designs may be highly unbalanced (rectangular). Below, we discuss two cases for which the maximum dimension can be reduced using the MIP formulation. In the first case, the maximum dimension is reduced without increasing the semiperimeter. In the second case, the maximum dimension is reduced while increasing the length of the semiperimeter.

In the first case, we observe that the 2-coloring solution of the induced bipartite subgraph G_B may not be unique. Consequently, different 2-coloring solutions for G_B may result in different maximum dimension but the same semiperimeter, which is illustrated with an example in Figure 6. If we remove the VH-node in Figure 6(a) and Figure 6(b), then we obtain two components in G_B. One possible 2-coloring solution is shown in Figure 6(a). The solution results in a semiperimeter S of seven and a maximum dimension D of five. However, we can relabel the top node from V to H, as illustrated in Figure 6(b). The resulting VH-labeling results in a more balanced solution with a semiperimeter S of seven but a maximum dimension of only four. In general, there exists an opportunity to reduce the maximum dimension when the induced subgraph G_B has two or more components. The MIP formulation is capable of finding the most balanced 2-coloring solution, which minimizes the maximum dimension.

(a) $|S| = 7$ and $|D| = 5$ (b) $|S| = 7$ and $|D| = 4$

Fig. 6. Unbalanced design to more balanced design by changing the 2-coloring

In the second case, we label additional nodes with VH such that we obtain a new, but smaller induced bipartite subgraph G_B. Remember from Eq 2 that the number of VH-nodes directly determines the semiperimeter. Thus, increasing the number of VH-nodes increases the semiperimeter. On the other hand, the smaller induced subgraph may provide opportunities to find VH-labeling solutions with an more even distribution of V and H labels. In Figure 7(a), the semiperimeter $S = 16$ and the maximum dimension $D = 10$. In Figure 7(b), both the root node and the terminal node are labeled with VH, such that we obtain a crossbar design with semiperimeter $S = 18$ and maximum dimension $D = 9$. Indeed, we have increased the semiperimeter S with the reward of finding a design with smaller maximum dimension D. The MIP formulation is intrinsically capable of performing this type of optimization.

(a) $S = 16$ and $D = 10$ (b) $S = 18$ and $D = 9$

Fig. 7. Unbalanced design to balanced design by adding VH-nodes

C. Scalability

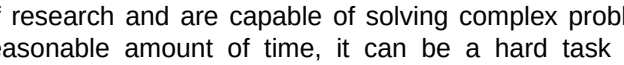
Finding an odd cycle transversal is NP-hard [28], [29]. Even though ILP solvers, such as CPLEX [30], are backed by years of research and are capable of solving complex problems in reasonable amount of time, it can be a hard task to find an optimal solution and to prove that the found solution is optimal. Finding an integer solution is even more challenging than finding a non-integer solution to the problem. Many ILP solvers allow the user to define a time limit for the task at hand. The solver will then return the best integer solution it has found and the best bound (non-integer solution). The relative gap is a measure of how good the integer solution is compared with the best bound. When the best integer solution and best bound converge, the ILP solver has found an optimal solution. Using the ILP formulation, one can always find a feasible solution: the trivial solution where all nodes are labeled VH. Of course, one is usually interested in alternative solutions with better properties.

VII. Extension to functions with multiple outputs

In Section VII-A, we will extend COMPACT from single-output functions to multi-output functions. A multi-output function can be represented by multiple ROBDDs or a single SBDD. In Section VII-B, additional constraints are introduced for the alignment of the inputs and the outputs.

A. Multiple ROBDDs vs single SBDDs

Previous work on flow-based computing for multi-input multi-output functions relied on splitting the function into many multi-input single-output functions. Next, each multi-input single-output function was converted into a ROBDD, merged by its terminal node 1 and mapped ta crossbar design. These individual crossbar designs are aligned along the diagonal, as illustrated in Figure 8(a). We observe that we can also directly convert the multi-input multi-output function into a SBDD [21], as illustrated in Figure 8(b). Next, the single SBDD can directly be mapped into a crossbar design. This may result in smaller crossbar designs as some parts of the single-output BDDs can be shared across multiple outputs.



(a) ROBDD (b) SBDD

Fig. 8. Multiple ROBDDs versus a single SBDD

We observe that in Figure 8 that the outputs are placed on any wordline. However, for flow-based computing, these wordlines will be permuted with the top-most nanowires. Also, the bottom-most wordlines of each sub-crossbar for each Boolean function will be placed to the bottom-most wordline through permutation.

B. Alignment of inputs and outputs

Flow-based computing relies on sensing resistors connected to wordlines. This requires each output of a multi-input multi-output function to be mapped to a wordline. Above this, leaf nodes of the SBDD will be mapped to the bottom-most

wordline. These requirements can be met by adding new constraints to the MIP formulation.

Let $T \subseteq U$ denote the set of terminal nodes and R let U denote the set of root nodes. For BDDs, $T = \{1\}$. Then we can add the following two constraints:

$$\begin{aligned} x_i^H &= 1; \quad \forall i \in R \\ x_i^H &= 1; \quad \forall i \in T \end{aligned} \tag{7}$$

The first and second constraint in Eq 7 force the root nodes and the terminal nodes, respectively, to be given at least a label H. This entails that the root nodes and terminal nodes can be mapped either horizontally when x_i^V is set to zero, or vertically, when x_i^V is also set to one.

VIII. Experimental evaluation

The COMPACT framework is implemented in Python and the experiments have been conducted on a machine with an Intel Core i9-9900X processor at 3.50GHz, 125GB of RAM memory, and Ubuntu 20.04 as operating system. We evaluate the effectiveness of the COMPACT framework using nine circuits of the ISCAS85 benchmark suite [31] and eight circuits of the EPFL control benchmarks [32]. A summary of the properties of the circuits is shown in Table I. The source code is publicly available on GitHub[1].

We evaluate COMPACT in terms of hardware utilization, power consumption, computation delay, and synthesis time. The alignment constraints are included by default. Hardware utilization is evaluated in terms of the crossbar dimensions, i.e. in terms of rows, columns, semiperimeter, maximum dimension, and area. The synthesis time is the run-time of COMPACT in the one-time initialization phase. Power consumption is proportional to the number of rows of the crossbar design and computation delay is the number of time-steps required to evaluate the Boolean function in the evaluation phase. The number of time steps is equal to the number of rows plus one. One time step per wordline is required to program the devices [33] and one time step is required to evaluate the Boolean function. Note that we have verified that all the crossbar designs are valid using SPICE simulations and the memristor model in [33].

[1]https://github.com/sventhijssen/compact

TABLE I
Overview of input circuits

Benchmark	Inputs	Outputs	Nodes	Edges
ISCAS85				
c432	36	7	1291	2578
c499	41	32	111146	222164
c880	60	26	4431	8858
c1355	41	32	111146	222164
c1908	33	25	28224	56348
c2670	233	140	6764	12970
c3540	50	22	59265	118442
c5315	178	123	14362	28232
c7552	207	108	90651	180870
EPFL control				
arbiter	256	129	25109	50214
cavlc	10	11	436	868
ctrl	7	26	89	174
dec	8	256	512	1020
i2c	147	142	1204	2404
int2 oat	11	7	159	314
priority	128	8	772	1540
router	60	30	219	434

First, we evaluate the influence of the user-defined parameter γ on the solution space in Section VIII-A. Then, for multi-output functions, we evaluate the crossbar designs for SBDDs and ROBDDs in Section VIII-B. We compare COMPACT with the state-of-the-art flow-based computing algorithm in Section VIII-D. An analysis of the scalability is made in Section VIII-C. We compare COMPACT with other in-memory computing paradigms in Section VIII-E.

A. Evaluation of γ

The user-defined parameter γ allows us to change the amount of pressure towards finding a solution with minimal semiperimeter or towards finding a solution with maximum dimension. In Table II, we evaluate COMPACT in terms of number of rows, columns, maximum dimension, semiperimeter, and synthesis time for different values of γ. The table only contains those benchmarks for which we find an optimal solution within three hours.

First, we compare COMPACT using $\gamma = 0$ and $\gamma = 0.5$. We observe that the maximum dimension D has decreased with 0.2% for $\gamma = 0$ compared with $\gamma = 0.5$. This is due to the fact that the maximum dimension D is being minimized by balancing the number of rows and the number of columns. However, the semiperimeter S for $\gamma = 0$ has increased with 3.6% compared with $\gamma = 0.5$. The latter can be explained due to the fact for $\gamma = 0$, the semiperimeter may be extended by introducing additional VH labels as long as the maximum dimension is not increased. It is expected that such solutions will be found because it is easier to resolve the connection constraints if additional VH labels are introduced. Consequently, it is not surprising that many of the crossbar designs synthesized with $\gamma = 0$ are square (or close to square). We observe that all except for the benchmark dec have an equal number of rows and columns. We conclude it is advantageous to set γ equal to 0.5 instead of 0.

Next, we compare COMPACT using $\gamma = 0.5$ with $\gamma = 1$. We observe that the normalized average of the maximum

TABLE II
SOLUTIONS IN TERMS OF NUMBER OF ROWS, COLUMNS, MAXIMUM DIMENSION , SEMIPERIMETER, AND SYNTHESIS TIME FOR DIFFERENT VALUES OF . THE BENCHMARKS ARE THOSE FOR WHICH AN OPTIMAL SOLUTION CAN BE FOUND WITHIN THREE HOURS

	= 0					= 0:5					= 1				
Benchmark	Rows (num)	Columns (num)	Max Dim (num)	Semi (num)	Time (h)	Rows (num)	Columns (num)	Max Dim (num)	Semi (num)	Time (h)	Rows (num)	Columns (num)	Max Dim (num)	Semi (num)	Time (h)
cavlc	233	233	233	466	0.0	236	225	236	461	0.0	239	220	239	459	0.0
ctrl	54	54	54	108	0.0	54	47	54	101	0.0	55	45	55	100	0.0
dec	341	255	341	596	0.0	341	170	341	511	0.0	341	170	341	511	0.0
i2c	658	658	658	1316	0.0	658	658	658	1316	0.0	677	627	677	1304	0.0
int2 oat	90	90	90	180	0.0	90	90	90	180	0.0	85	95	95	180	0.0
priority	449	449	449	898	0.0	449	449	449	898	0.0	440	458	458	898	0.0
router	121	121	121	242	0.0	120	121	121	241	0.0	122	119	122	241	0.0
Normalized			0.998	1.036				1.000	1.000				1.021	0.997	

TABLE III
SOLUTIONS IN TERMS OF NUMBER OF ROWS, COLUMNS, MAXIMUM DIMENSION , SEMIPERIMETER, AND SYNTHESIS TIME FOR MULTIPLE ROBDDS AND A SINGLE SBDD FOR = 0:5.

Benchmark	ROBDD						SBDD					
	Nodes (num)	Rows (num)	Cols (num)	Max Dim (num)	Semi (num)	Time (h)	Nodes (num)	Rows (num)	Cols (num)	Max Dim (num)	Semi (num)	Time (h)
ISCAS85												
c432	1414	833	833	833	1666	3.00	1291	528	528	528	1056	3.00
c499	111146	67639	67639	67639	135278	10.30	111146	67639	65981	67639	133620	10.30
c880	5776	3339	3339	3339	6678	3.03	4431	1883	1798	1883	3681	3.01
c1355	111146	65812	65812	65812	131624	10.49	111146	65812	64209	65812	130021	10.49
c1908	30605	16000	16000	16000	32000	3.41	30605	16000	16061	16061	32061	3.41
c2670	8250	4126	4126	4126	8252	3.06	8250	4126	4153	4153	8279	3.06
c3540	59265	32598	32598	32598	65196	5.14	59265	32598	32590	32598	65188	5.14
c5315	15454	7925	7925	7925	15850	2.43	15454	7925	7926	7926	15851	2.43
c7552	33983	18440	18440	18440	36880	3.55	33983	18440	18438	18440	36878	3.55
EPFL control												
arbiter	42092	24914	24914	24914	49828	4.35	25109	10301	10310	10310	20611	3.55
cavlc	602	311	311	311	622	0.02	436	199	210	210	409	0.02
ctrl	243	101	101	101	202	0.01	89	34	41	41	75	0.00
dec	2560	1024	1024	1024	2048	0.00	512	170	341	341	511	0.00
i2c	2698	1312	1312	1312	2624	0.06	1204	545	545	545	1090	0.02
int2 oat	227	121	121	121	242	0.00	159	68	68	68	136	0.01
priority	913	512	512	512	1024	0.01	772	322	322	322	644	0.02
router	283	127	127	127	254	0.01	219	99	98	99	197	0.01
Normalized	1.00	1.00	1.00	1.00	1.00	1.00	0.78	0.71	0.73	0.73	0.72	0.88

dimension D has increased with 2:1% compared with the solutions for = 0:5. On the other hand, the normalized semiperimeter S has decreased with 0:3%. This is what we would expect, considering that = 1 only minimizes S while = 0:5 minimizes a weighted combination of the semiperimeter S and the maximum dimension D. Now we analyze the results on a few circuits in detail. For the circuits int2 oat, priority, and router, we observe that = 0:5 reduces the maximum dimension without a penalty on the semiperimeter. For example, the maximum dimension is reduced from 458 to 449 on priority. This stems from that the MIP formulation determines more balanced 2-coloring solutions without introducing any additional VH-nodes. For the other circuits, except for dec, we observe that the maximum dimension is reduced at the expense of increasing the length of the semiperimeter. In particular, the maximum dimension is reduced with 1, and 19 at the expense of increasing the semiperimeter with 2, 1, and 12, respectively.

Based on the results in Table II, we conclude that the best crossbar designs are obtained using = 0:5. It seems advantageous to increase the average semiperimeter with about 3% in order to reduce the the average maximum dimension with about 2%. Therefore, we set the default value for to be 0:5 in the remainder of the experimental evaluation.

For the circuits cavlc and int2 oat, we sweep between 0 and 1 and examine all non-dominated crossbar designs in Figure 9. A crossbar design is non-dominated if no other crossbar design is obtained with a smaller number of rows and a smaller number of columns. The number of rows and columns is shown on the x-axis and y-axis, respectively. For the circuit cavlc, we find the following non-dominated solutions: (233; 233), (233; 232), (234; 229), (236; 225), (238; 221), and (239; 220). On the circuit int2 oat, we found the non-dominated solutions (90; 90), (90; 89), and (92; 87).

B. Evaluation of multiple ROBDDs vs single SBDD

In Table III, we compare the hardware utilization of COMPACT for both multiple ROBDDs and a single SBDD. We observe that the number of nodes in the SBDD is smaller than or equal to the total number of nodes in the merged ROBDDs. This leads to crossbar designs with fewer rows and columns, and thus smaller semiperimeter and area for COMPACT using SBDDs. COMPACT using SBDDs reduces the number of

Fig. 9. Dimensions of different crossbar designs for varying values of

nodes by 22% on the average. The resulting number of rows and columns is reduced by 29% and 27% on the average. The maximum dimension D and the semiperimeter S are respectively reduced by 27% and 28%. Above this, the synthesis time decreases with 12%. We conclude that COMPACT using SBDDs results in much smaller crossbar designs and we will use this BDD type in the remainder of the experiments.

C. Scalability

In Table IV, we observe that the synthesis time for COMPACT with $= 0.5$ is approximately 2650X the required synthesis time for the work in [16]. This difference in synthesis time stems from the fact that the mapping algorithm in [16] is linear in the number of nodes of the undirected graph G whereas finding an odd cycle transversal is NP-hard as explained in Section VI-C.

In Figure 10, we illustrate how the ILP solver converges to an optimal solution for the benchmark i2c for $= 0.5$. The best integer solution, the best bound and the relative gap between the best integer and best bound are given in terms of the elapsed time. We observe that the best integer decreases over time whereas the bound increases. When both the bound and integer solution have converged, then we the ILP solver has found an optimal solution. Further we provide the relative gap between both values. Initially, the relative gap is nearly 100% We observe that the best integer solution makes several jumps to finally converge with the best non-integer solution have converged, closing the relative gap between both.

Fig. 10. Best integer solution, best bound, and relative gap for the benchmark i2c for $= 0.5$ in terms of elapsed time

In Figure 11, we provide the relative gap for those benchmarks for which we have not found an optimal solution within three hours. We observe that the relative gap for c499, c1355 and arbiter and relatively large compared with the other benchmarks. Some structures are inherently complex, which requires the ILP solver more time to find better solutions.

Fig. 11. Relative gap for the benchmarks for which no optimal solution was found within three hours

D. Comparison with previous work on flow-based computing

In this section, we compare COMPACT with the state-of-the-art flow-based computing algorithm in [16]. We evaluate the performance in terms of hardware utilization in Table IV. The power consumption and computation delay are evaluated in Figure 12.

The number of BDD nodes, the number of rows, columns, semiperimeter and area for each of the techniques is shown in Table IV. It can be observed that the algorithm in [16] is capable of mapping all the input circuits into valid crossbar representations. The semiperimeter is approximately $1.99n$, where n is the number of BDD nodes. The run-time is less than eight minutes for all circuits.

Compared with the algorithm in [16], COMPACT with $= 0.5$ reduces the number of rows, columns, maximum dimension, semiperimeter and area with 61%, 77%, 85%, 55%, and 89%, respectively. Smaller crossbar designs are obtained due to the fact that most BDD nodes are only mapped to a single wordline or bitline, whereas in the previous work all nodes are mapped to at least one wordline and one bitline. The semiperimeter for COMPACT using $= 0.5$ is approximately $1.11n$, which demonstrates that only 11% of the nodes in the BDD are labeled VH and are mapped to both a wordline and bitline.

(a) Power consumption (b) Computation delay

Fig. 12. Comparison of power consumption and computation delay for COMPACT with $= 0.5$ and flow-based computing synthesis algorithm by Chakraborty et al. [16]

Figure 12 shows the normalized power and computation delay for the two methods across all the circuits. It can be

TABLE IV
COMPARISON OF FLOW-BASED COMPUTING ALGORITHMS IN TERMS OF NUMBER OF BDD NODES, ROWS, COLUMNS, SEMIPERIMETER, AREA AND SYNTHESIS TIME.

Benchmark	Chakraborty et al. [16]								COMPACT for $\gamma = 0.5$							
	Nodes (num)	Edges (num)	Rows (num)	Columns (num)	Max Dim (num)	Semi (num)	Area (num)	Time (h)	Nodes (num)	Edges (num)	Rows (num)	Columns (num)	Max Dim (num)	Semi (num)	Area (num)	Time (h)
ISCAS85																
c432	1414	2800	1414	2800	4214	2800	3959200	0.00	1291	2578	762	762	762	1524	580644	3.00
c499	111146	222164	111146	222164	333310	222164	24692639944	0.00	111146	222164	65981	67639	67639	133620	4462888859	10.30
c880	5776	11448	5776	11448	17224	11448	66123648	0.02	4431	8858	2547	2632	2632	5179	6703704	3.01
c1355	111146	222164	111146	222164	333310	222164	24692639944	0.00	111146	222164	64209	65812	65812	130021	4225722708	10.49
c1908	30605	61110	30605	61110	91715	61110	1870271550	0.00	30605	61110	16061	16000	16061	32061	256976000	3.41
c2670	8250	15941	8250	15941	24191	15941	131513250	0.00	8250	15941	4153	4126	4153	8279	17135278	3.06
c3540	59265	118442	59265	118442	177707	118442	7019465130	0.00	59265	118442	32590	32598	32598	65188	1062368820	5.14
c5315	15454	30416	15454	30416	45870	30416	470048864	0.12	15454	30416	7926	7925	7926	15851	62813550	2.43
c7552	33983	67534	33983	67534	101517	67534	2295007922	0.00	33983	67534	18438	18440	18440	36878	339996720	3.55
EPFL control																
arbiter	42092	83668	42092	83668	125760	83668	3521753456	0.00	25109	50214	15030	15035	15035	30065	225976050	3.25
cavlc	602	1160	602	1160	1762	1160	698320	0.00	436	868	236	225	236	461	53100	0.02
ctrl	241	380	241	380	621	380	91580	0.00	89	174	54	47	54	101	2538	0.00
dec	2560	4096	2560	4096	6656	4096	10485760	0.01	512	1020	341	170	341	511	57970	0.00
i2c	2696	4826	2696	4826	7522	4826	13010896	0.00	1204	2404	658	658	658	1316	432964	0.02
int2float	227	426	227	426	653	426	96702	0.00	159	314	90	90	90	180	8100	0.01
priority	913	1794	913	1794	2707	1794	1637922	0.00	772	1540	449	449	449	898	201601	0.02
router	256	446	256	446	702	446	114176	0.00	219	434	120	121	121	241	14520	0.01
Normalized	1.00	1.00	1.00	1.00	1.00	1.00	1.00	**1.00**	**0.79**	**0.81**	**0.44**	**0.23**	**0.15**	**0.45**	**0.11**	2653.22

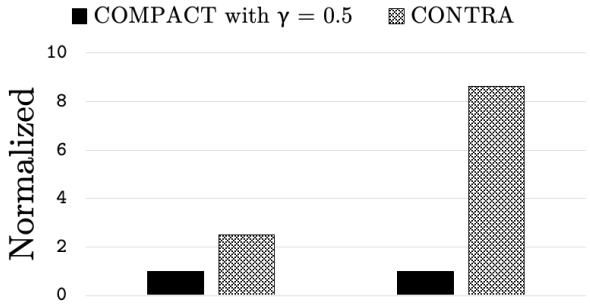
observed that COMPACT with = 0:5 requires less power consumption than the algorithm proposed in [16]. This stems from that the number of memristors that are required to be programmed is equal to the number of edges in the BDD. A single SBDD in COMPACT can be smaller than the set of individual ROBDDs in [16], resulting in a reduction of 19% on average. Compared with [16], COMPACT with = 0:5 reduces the computation delay with 56%. This stems from that crossbar designs with fewer rows are synthesized, which results in that it takes shorter time to program the memristors in the crossbar based on the Boolean input variables.

E. Comparison of with other in-memory computing paradigms

In this section, we compare COMPACT with in-memory computing based on MAGIC. More specifically, we compare our work with the state-of-the-art mapping method CONTRA [34]. In Figure 13, we compare the power consumption, and the computational delay for both CONTRA and COMPACT with = 0:5. The user-defined parameters for CONTRA are k = 4, spacing= 6, and a crossbar dimension of 128 128 where k is the number of inputs for the LUTs, and the spacing denotes the number of rows between multiple LUTs in the crossbar. We have chosen k = 4, as it is reported that the delay is lowest for k = 4. The spacing and crossbar dimension were chosen as these were the most common values.

For CONTRA, both the power consumption and computational delay are expressed in terms of the number of the operations (INPUT, COPY, ...) where each operation is considered a write operation. For COMPACT, the power consumption is expressed in terms of the number of active memristors, i.e. the number of literals assigned to the memristors; worst-case, all memristors have to be programmed to a high resistive state, requiring most power. The computational delay is expressed in the number of rows; worst-case all rows have to be reprogrammed using the method in [33].

Figure 13 shows that the normalized power consumption for COMPACT with = 0:5 is 55% lower than the power consumption for CONTRA for the EPFL control benchmarks. Here, we only compare COMPACT with CONTRA the EPFL control benchmarks as the ISCAS85 benchmarks are arithmetic circuits and BDDs do not scale well for these types of circuits. The improvements stem from that it is difficult to achieve high parallelism within the MAGIC style. While it is possible to evaluate many "logic gates" in a single time step, the subsequent time steps will be spent attempting to realign the data to perform a highly parallel operation again. COMPACT with = 0:5 also improves the computational delay with 87%. The computational delay is 8:65X higher for CONTRA due to the high number of computational steps.



(a) Power consumption (b) Computation delay

Fig. 13. Comparison of power consumption and computation delay for COMPACT with $\gamma = 0.5$ and CONTRA with $k = 4$, spacing= 6, and dimensions 128×128 on the EPFL control benchmarks

IX. CONCLUSION AND FUTURE WORK

In this paper, we presented COMPACT for mapping Boolean functions to crossbar representations for flow-based in-memory computing. By utilizing an analogy between a BDD and a crossbar, COMPACT with = 0:5 reduces the semiperimeter by 55% and the maximum dimension by 85% compared with previous work on flow-based computing. We have introduced a weighted objective of the semiperimeter and the maximum dimension to find small valid crossbar designs. A MIP formulation was proposed to minimize the weighted objective. We have extended the framework from multi-input functions using ROBDDs to multi-output functions using SBDDs. Above this, alignment constraints were introduced to map the inputs and the outputs to wordlines.

ACKNOWLEDGMENT

The authors acknowledge support from the National Science Foundation awards #2113307, the DARPA cooperative agreement #HR00112020002, and ONR grant #N000142112332. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

[1] J. Von Neumann, "First draft of a report on the edvac," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993.

[2] J. Backus, "Can programming be liberated from the von neumann style?: A functional style and its algebra of programs," *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, 1978.

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

[4] 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.

[5] 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.

[6] 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.

[7] S. K. Jha, D. E. Rodriguez, J. E. Van Nostrand, and A. Velasquez, "Computation of boolean formulas using sneak paths in crossbar computing," Apr. 19 2016. US Patent 9,319,047.

[8] S. Shirinzadeh, M. Soeken, and R. Drechsler, "Multi-objective bdd optimization for rram based circuit design," in *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 1–6, IEEE, 2016.

[9] E. Lehtonen, J. Poikonen, and M. Laiho, "Implication logic synthesis methods for memristors," in *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2441–2444, IEEE, 2012.

[10] N. Talati, R. Ben-Hur, N. Wald, A. Haj-Ali, J. Reuben, and S. Kvatinsky, "mmpu—a real processing-in-memory architecture to combat the von neumann bottleneck," in *Applications of Emerging Memory Technology*, pp. 191–213, Springer, 2020.

[11] A. Velasquez and S. K. Jha, "Parallel computing using memristive crossbar networks: Nullifying the processor-memory bottleneck," in *2014 9th International Design and Test Symposium (IDT)*, pp. 147–152, IEEE, 2014.

[12] D. Chakraborty, S. Raj, S. L. Fernandes, and S. K. Jha, "Input-aware flow-based computing on memristor crossbars with applications to edge detection," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 580–591, 2019.

[13] A. Velasquez and S. K. Jha, "Fault-tolerant in-memory crossbar computing using quantified constraint solving," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*, pp. 101–108, IEEE, 2015.

[14] 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 International Symposium on Circuits and Systems (ISCAS)*, pp. 1870–1873, IEEE, 2016.

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

[16] D. Chakraborty and S. K. Jha, "Automated synthesis of compact crossbars for sneak-path based in-memory computing," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 770–775, IEEE, 2017.

[17] A. U. Hassen, D. Chakraborty, and S. K. Jha, "Free binary decision diagram-based synthesis of compact crossbars for in-memory computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 5, pp. 622–626, 2018.

[18] D. Chakraborty, S. Raj, J. C. Gutierrez, T. Thomas, and S. K. Jha, "In-memory execution of compute kernels using flow-based memristive crossbar computing," in *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–6, IEEE, 2017.

[19] A. U. Hassen, S. A. Khokhar, H. A. Butt, and S. K. Jha, "Free bdd based cad of compact memristor crossbars for in-memory computing," in *2018 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pp. 1–7, IEEE, 2018.

[20] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. 100, no. 8, pp. 677–691, 1986.

[21] S.-i. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagram with attributed edges for efficient boolean function manipulation," in *ACM/IEEE DAC 1990*, pp. 52–57, IEEE, 1990.

[22] C. Xu, D. Niu, N. Muralimanohar, R. Balasubramonian, T. Zhang, S. Yu, and Y. Xie, "Overcoming the challenges of crossbar resistive memory architectures," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 476–488, IEEE, 2015.

[23] A. Berkeley, "A system for sequential synthesis and verification," 2009.

[24] D. B. Strukov and R. S. Williams, "Four-dimensional address topology for circuits with stacked multilayer crossbar arrays," *Proceedings of the National Academy of Sciences*, vol. 106, no. 48, pp. 20155–20158, 2009.

[25] D. B. West, *Introduction to Graph Theory*, vol. 2. Prentice hall Upper Saddle River, NJ, 1996.

[26] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*, vol. 4. Springer, 2015.

[27] V. V. Vazirani, *Approximation Algorithms*. Springer Science & Business Media, 2013.

[28] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*, pp. 85–103, Springer, 1972.

[29] C. Lund and M. Yannakakis, "The approximation of maximum subgraph problems," in *International Colloquium on Automata, Languages, and Programming*, pp. 40–51, Springer, 1993.

[30] "Cplex optimizer."

[31] F. Brglez, P. Pownall, and R. Hum, "Accelerated atpg and fault grading via testability analysis," in *Proceedings of IEEE Int. Symposium on Circuits and Systems*, pp. 695–698, 1985.

[32] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, no. CONF, 2015.

[33] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Memristor spice model and crossbar simulation based on devices with nanosecond switching time," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, IEEE, 2013.

[34] D. Bhattacharjee, A. Chattopadhyay, S. Dutta, R. Ronen, and S. Kvatinsky, "Contra: area-constrained technology mapping framework for memristive memory processing unit," in *Proceedings of the 39th International Conference on Computer-Aided Design*, pp. 1–9, 2020.