Implementing Boolean Function by Ternary Content Addressable Memory with Approximate Match

Jian Shi[1], Weikang Qian[1,2,*]
[1]University of Michigan-SJTU Joint Institute and [2]MoE Key Lab of AI, Shanghai Jiao Tong University, China
Email: {timeshi, qianwk}@sjtu.edu.cn; *corresponding author

***Abstract*—Ternary content addressable memory (TCAM) is a widely used component for high-speed lookup operation. In this work, we advocate a novel use of TCAM, *i.e.*, for implementing a Boolean function. We further leverage approximate match to reduce the resource usage. To achieve this, two extra columns are added to the TCAM-based architecture. The experimental results show that to support the implementation of any 4-input Boolean functions, the proposed architecture can reduce 37.5% rows and 12.5% bit cells over the conventional architecture.**

***Index Terms*—Boolean function, TCAM, approximate match**

I. Introduction

Ternary content addressable memory (TCAM) is a special memory that can be searched by content instead of location. It is widely used in high-speed search operation such as data compression, network router, and image processing [1].

The design of efficient TCAM architecture has attracted much attention recently. For example, Chang *et al.* introduce a TCAM structure with lower power consumption in peripheral circuits [2]. Ghofrani *et al.* propose an approximate match technique for TCAM built with nonvolatile devices, which enables the match of more inputs [3]. However, all existing architectures are only used as a lookup table for storing frequently-used patterns. In this work, we advocate the use of TCAM to implement a Boolean function. The basic idea is to represent a function as an optimized sum-of-product (SOP) expression and then use TCAM to store the product terms in the expression. Furthermore, we explore the approximate match technique to reduce the total number of bit cells in a TCAM-based architecture. To achieve this, two extra columns are added to the TCAM-based architecture.

II. Background

A TCAM compares an input pattern with the stored patterns and activates the *matchline* (*ML*) of the matching pattern. The left part of Fig. 1 shows a TCAM design. It consists of k rows, each storing an n-bit word and associated with a ML. The input pattern is fed through the *searchlines* (*SLs*). If it matches a word, the corresponding ML is activated.

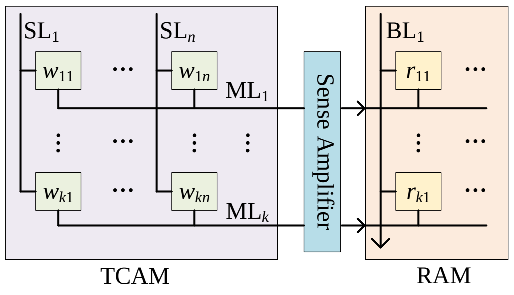

Fig. 1: An n-input TCAM associated with a RAM.

The TCAM can be associated with a RAM shown in the right part of Fig. 1. Once the i-th ML is activated, the RAM outputs the value stored in its i-th row through the *bitlines* (*BLs*). In fact, the i-th BL performs an OR operation on its bit cells enabled by the corresponding MLs, *i.e.*,

$$\mathrm{BL}_i = r_{1i} \cdot \mathrm{ML}_1 + \cdots + r_{ki} \cdot \mathrm{ML}_k, \tag{1}$$

In this work, we consider a nonvolatile TCAM implementation using 2 transistors and 2 memristors (2T-2R) in each bit cell [4]. For a 2T-2R TCAM, the MLs are first precharged. If the input pattern has bit differences over the stored word, the corresponding ML is discharged by the mismatch bit cells in the row. The number of mismatch bit cells equals the *Hamming distance* (*HD*) between the input pattern and the stored word. The larger the HD, the faster the ML is discharged. Fig. 2 shows how the voltage of ML drops with time for different HDs between the input pattern and the stored word. Clearly, the voltage decrease of the 1-HD case is much slower than the other HD cases. Ghofrani *et al.* proposed to shorten the sampling period for approximate computing [3]. In this situation, the voltage of the ML for the 1-HD case is still high when sampled, and consequently, an input pattern that is 1 HD from the stored word can be treated as matching with the stored word. This technique is called *1-HD match* in TCAM. In contrast, an input pattern that is at least 2 HDs from the stored word cannot activate the ML of the word.

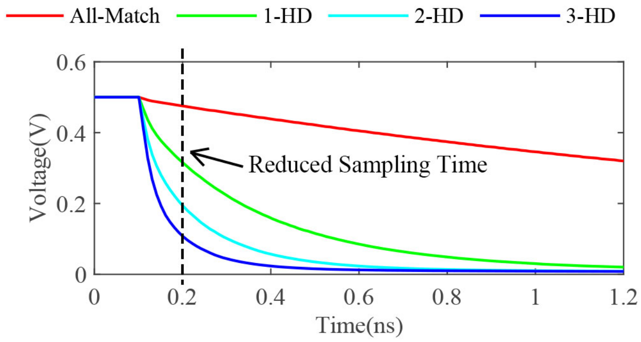

Fig. 2: The ML voltages under different HD cases.

III. Methodology

In this work, we advocate the use of TCAM to implement a Boolean function. Consider the Boolean function represented by a Karnaugh map shown in Table I(a). Its simplest SOP expression is

$$f([a,b,c,d]) = \bar{a} \cdot b \cdot \bar{c} + \bar{a} \cdot b \cdot d + \bar{a} \cdot \bar{c} \cdot d + b \cdot \bar{c} \cdot d \tag{2}$$

TABLE I: The Karnaugh maps of two 4-input Boolean functions.

(a)

$cd\backslash ab$	00	01	11	10
00	0	**1**	0	0
01	**1**	**1**	**1**	0
11	0	**1**	0	0
10	0	0	0	0

(b)

$cd\backslash ab$	00	01	11	10
00	0	0	**1**	0
01	0	**1**	**1**	**1**
11	0	0	**1**	0
10	**1**	0	0	0

We can implement the SOP by a TCAM-based architecture shown in Fig. 3. The TCAM part has 4 rows storing the product terms in the SOP, *i.e.*, $[0, 1, 0, X]$, $[0, 1, X, 1]$, $[0, X, 0, 1]$, and $[X, 1, 0, 1]$, where X represents an input don't care bit. For the RAM part, each bit cell stores 1. With such a setup, by Eq. (1), the final output equals the given Boolean function.

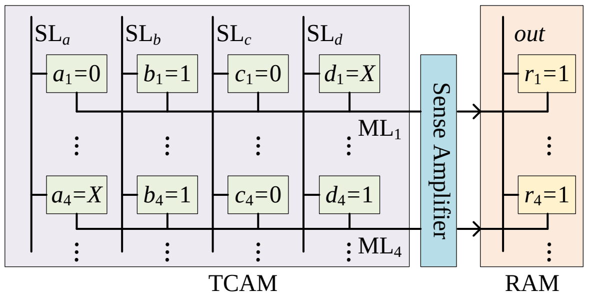

Fig. 3: A TCAM-based architecture to implement the Boolean function in Eq. (2).

However, by exploiting the 1-HD match, we can reduce the number of needed rows to 1. Specifically, we only need to store the word $[0, 1, 0, 1]$ in the TCAM and store 1 in the bit cell of the associated RAM. For any input pattern in the ON set of the Boolean function f shown in Table I(a), since it either equals $[0, 1, 0, 1]$ or is 1 HD from $[0, 1, 0, 1]$, by 1-HD match, it activates the ML of the stored word $[0, 1, 0, 1]$. Consequently, the final output is 1. For any input pattern in the OFF set, since it is at least 2 HDs from $[0, 1, 0, 1]$, it cannot activate the ML, leading to a final output of 0.

The above example shows that if 5 on-set input patterns form a cross-like pattern in the Karnaugh map, as shown in Table I(a), then by exploiting the 1-HD match, we can reduce the number of rows stored in the TCAM. We call this technique *cross pattern-enabled row reduction*. Inspired by the above example, we try to exploit 1-HD match and cross pattern-enabled row reduction to minimize the hardware cost.

However, if the 1-HD match is enabled in the TCAM, some Boolean functions cannot be correctly implemented. Consider the one shown in Table I(b). To implement it, the pattern $[0, 0, 1, 0]$ or its 1-HD neighbour should be stored in the TCAM. Then, some OFF set input patterns can also activate the ML. For example, if the pattern $[0, 0, 1, 0]$ is stored, then the input pattern $[0, 0, 1, 1]$ can also activate the ML of the row, causing a wrong output value of 1. To implement an arbitrary Boolean function, we propose a solution in Section III-A by introducing an extra column in the TCAM. Furthermore, we notice that for some cases, it is impossible to exploit cross pattern-enabled row reduction. To address this, we propose an improved design with an extra column in the RAM in Section III-B.

A. Extra Column in TCAM for Arbitrary Boolean Function Implementation

To implement an arbitrary Boolean function, we introduce an extra column in the TCAM. Then, each row in the TCAM has an extra bit cell ε. When we want to get the output of an input pattern, the input pattern is appended with an extra bit $\delta = 0$ and fed into the TCAM. We apply the following rules to configure the TCAM.

1) When a pattern v is in the ON set of the Boolean function and some of its 1-HD neighbours are in the OFF set, the pattern is stored in a row r as usual with the extra bit cell ε of the row storing 1. By this configuration, when we look for the output of the input pattern v, the input to the TCAM is v appended with a 0. In this case, the input and the stored word only have 1 bit difference, which occurs at the extra column. Therefore, the input activates the ML of row r, and the TCAM outputs a 1, which is the correct output for the input pattern v. When we look for the output of a 1-HD neighbour u of v in the OFF set, the input to the TCAM is u appended with a 0. In this case, the input is 2 HDs from the word stored at row r. Thus, it does not activate the ML of row r, and the TCAM outputs a 0, which is the correct output for the input pattern u.
2) When a pattern v and all of its 1-HD neighbours belong to the ON set, the pattern is stored in a row r as usual with the extra bit cell ε of the row storing 0. By this configuration, when we look for any input pattern u within 1 HD from v, the input to the TCAM is u appended with a 0, which is within 1 HD from the word stored at row r. Thus, the ML of row r is activated, and the TCAM outputs a 1, which is the correct output for the input pattern u.

With the proposed technique, only two rows are needed in the TCAM to implement the Boolean function shown in Table I(b). The corresponding stored patterns are

$$[a, b, c, d, \varepsilon] = [0, 0, 1, 0, 1] \text{ and } [a, b, c, d, \varepsilon] = [1, 1, 0, 1, 0],$$

where the first row can only be activated by the input pattern $[0, 0, 1, 0]$, and the second row can be activated by the remaining ON set input patterns.

B. Extra Column in RAM for Row Reduction

Although by introducing an extra column in the TCAM, we can implement an arbitrary Boolean function, a target function sometimes has no cross pattern that enables row reduction, *e.g.*, the Boolean function in Table II(a).

To maximally exploit the cross pattern-enabled row reduction, we propose an improved architecture shown in Fig. 4, where we add an extra column to the RAM part. Furthermore, the final output out_c is the XOR of the output out' of the original column in the RAM and the output σ of the extra column in the RAM, *i.e.*, $out_c = out' \oplus \sigma$. We call the architecture *approximate match-based TCAM* (*AM-TCAM*).

With the new architecture, we can exploit the cross pattern-enabled row reduction. Consider the Boolean function in

TABLE II: A 4-input Boolean function: (a) its Karnaugh map; (b) the configuration of AM-TCAM to implement it.

(a)

$cd\backslash ab$	00	01	11	10
00	0	**1**	0	0
01	**1**	**1**	**0**	0
11	0	**1**	0	0
10	0	0	0	0

(b)

TCAM					RAM	
a	b	c	d	ε	out'	σ
0	1	0	1	0	1	0
1	1	0	1	1	1	1

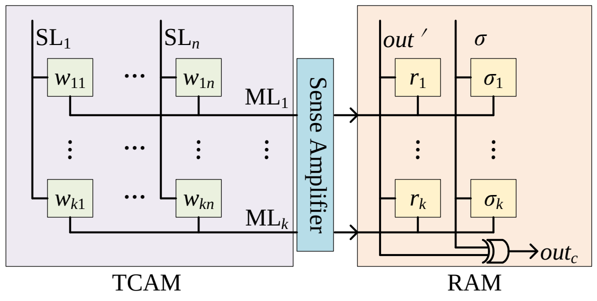

Fig. 4: The proposed AM-TCAM architecture.

Table II(a) again. The basic idea is to first treat the output of the input pattern $[1, 1, 0, 1]$ as 1, which leads to a cross pattern that enables row reduction, and then correct the wrong output value of the pattern $[1, 1, 0, 1]$. To achieve this, we configure the TCAM and the RAM as shown in Table II(b). Specifically, by treating the output of the input pattern $[1, 1, 0, 1]$ as 1, we identify a cross pattern and hence, can reduce the number of rows by storing the center of the cross pattern, *i.e.*, $[0, 1, 0, 1]$, in a row of the TCAM with the extra bit cell ε set as 0. For this row, the original and extra RAM bit cells store 1 and 0, respectively. To correct the wrong output value for the pattern $[1, 1, 0, 1]$, the pattern is stored in another row of the TCAM with the extra bit cell ε set as 1. The corresponding original and extra RAM bit cells both store 1. Under such a configuration, we have:

1) For the ON set input patterns $[0, 1, 0, 0]$, $[0, 0, 0, 1]$, $[0, 1, 0, 1]$, and $[0, 1, 1, 1]$, the first row in Table II(b) is activated due to at most 1 HD from the input, while the second row is not due to at least 2 HDs from the input. By Eq. (1), the RAM outputs are $out' = 1$ and $\sigma = 0$. Therefore, the final output is $out_c = out' \oplus \sigma = 1$.
2) For the input pattern $[1, 1, 0, 1]$, both rows in Table II(b) are activated due to 1-HD match. By Eq. (1), the RAM outputs are $out' = 1$ and $\sigma = 1$. Therefore, the final output is $out_c = out' \oplus \sigma = 0$.

Thus, AM-TCAM realizes the target Boolean function. Furthermore, it needs 10 bit cells in the TCAM and 4 bit cells in the RAM. In contrast, the *conventional architecture* without exploiting 1-HD match needs to store 3 words, *i.e.*, $[0, 1, 0, X]$, $[0, 1, X, 1]$, and $[0, X, 0, 1]$, leading to 12 bit cells in the TCAM and 3 bit cells in the RAM. This example shows the benefit of AM-TCAM in resource usage reduction.

IV. Experimental Results

In this section, we compared the resource usage between AM-TCAM and the conventional architecture, measured by the total number of bit cells in the TCAM and the RAM, for 4-input Boolean functions. It can be shown that all Boolean functions in the same negation-permutation-negation (NPN) equivalence class need the same total number of bit cells. Thus, for each NPN equivalence class, we choose one function in it as the test case, which is the one with the minimum ON set size in the class. All the test cases are simplified by ESPRESSO [5]. We find that the following function needs the maximum number of rows for the conventional architecture:

$$f([a, b, c, d]) = a \oplus b \oplus c \oplus d. \tag{3}$$

Indeed, it needs 8 rows. However, using AM-TCAM, only 4 rows are required, and its configuration is shown in Table III.

TABLE III: The configuration of AM-TCAM to implement the Boolean function in Eq. (3).

TCAM					RAM	
a	b	c	d	ε	out'	σ
0	0	1	1	0	1	0
0	0	1	1	1	1	1
1	1	0	0	0	1	0
1	1	0	0	1	1	1

Moreover, we manually configure AM-TCAM for the selected representative functions of all the NPN equivalence classes. We want to obtain the maximum number of rows needed over all the functions, since an architecture with the number of rows equal to that maximum value can implement any 4-input Boolean function. We find that the maximum number of rows needed is 5. Thus, to support the implementation of any 4-input Boolean function, the total number of cells in AM-TCAM is $5 \times 7 = 35$. The conventional architecture needs 8 rows to support the implementation of any 4-input function, and the total number of cells it contains is $8 \times 5 = 40$. Therefore, AM-TCAM reduces 37.5% rows and 12.5% bit cells over the conventional architecture.

V. Conclusion

This work proposes to implement Boolean functions by a TCAM-based architecture. To reduce the resource usage, it exploits 1-HD match and contains an extra column in the TCAM part and an extra column in the RAM part. The proposed architecture can realize single-output Boolean functions with a limited number of inputs. Thus, it has the potential to replace the conventional lookup tables used in FPGA. Currently, the configuration of the proposed architecture for a given Boolean function is done manually. Our future work will develop an algorithm for automatic configuration.

References

[1] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *JSSC*, vol. 41, no. 3, pp. 712–727, 2006.

[2] M.-F. Chang *et al.*, "A 3T1R nonvolatile TCAM using MLC ReRAM for frequent-off instant-on filters in IoT and big-data processing," *JSSC*, vol. 52, no. 6, pp. 1664–1679, 2017.

[3] A. Ghofrani *et al.*, "Associative memristive memory for approximate computing in GPUs," *JETCAS*, vol. 6, no. 2, pp. 222–234, 2016.

[4] J. Li *et al.*, "1 Mb 0.41 µm² 2T-2R cell nonvolatile TCAM with two-bit encoding and clocked self-referenced sensing," *JSSC*, vol. 49, no. 4, pp. 896–907, 2014.

[5] R. K. Brayton *et al.*, *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.