Patterned Cells for Phase Change Memories

Anxiao (Andrew) Jiang
Computer Sci. and Eng.
Texas A&M University
College Station, TX 77843
ajiang@cse.tamu.edu

Hongchao Zhou
Electrical Engineering
Caltech
Pasadena, CA 91125
hzhou@caltech.edu

Zhiying Wang
Electrical Engineering
Caltech
Pasadena, CA 91125
zhiying@caltech.edu

Jehoshua Bruck
Electrical Engineering
Caltech
Pasadena, CA 91125
bruck@caltech.edu

***Abstract*—Phase-change memory (PCM) is an emerging nonvolatile memory technology that promises very high performance. It currently uses discrete cell levels to represent data, controlled by a single amorphous/crystalline domain in a cell. To improve data density, more levels per cell are needed. There exist a number of challenges, including cell programming noise, drifting of cell levels, and the high power requirement for cell programming.**

In this paper, we present a new cell structure called *patterned cell*, and explore its data representation schemes. Multiple domains per cell are used, and their connectivity is used to store data. We analyze its storage capacity, and study its error-correction capability and the construction of error-control codes.

I. Introduction

Phase-change memory (PCM) is an important emerging nonvolatile memory (NVM) technology that promises high performance. It uses chalcogenide glass as cells, which has two stable states: amorphous and crystalline [2]. The amorphous state has very high electrical resistance, and the crystalline state has low resistance. Intermediate states, called partially crystalline states, can also exist. High temperatures induced by electrical currents are used to switch the state of a portion of the cell, which is called a *domain*. By quantizing cell resistance into multiple discrete levels, one or more bits per cell can be stored. Currently, four-level cells have been developed. To improve data density, more levels are needed [2].

The current multi-level cell (MLC) approach faces a number of challenges, including cell-programming noise, cell-level drifting, and high power consumption [2], [4]. It is difficult to program cell levels accurately due to cell heterogeneity and noise. The cell levels can drift away significantly after they are programmed, making it even harder to control their accuracy. And the high power requirement for cell programming is hindering PCM's application in mobile devices [4].

In this paper, we explore a new cell structure and its data representation scheme. In the new structure, called *patterned cells*, multiple domains per cell are used. An example is shown in Fig. 1, where two or four domains exist in a cell, whose states are independently controlled by their respective bottom electrodes. (The state of a domain is switched by the current between the bottom and top electrodes. We assume that the PCM layer is sufficiently thin such that changing a domain to the crystalline state, which is called the SET operation and requires a lower temperature/current, will not affect its neighboring domains.) The base of a cell is in the amorphous state, while every domain can be switched to the crystalline state. (To change domains back to amorphous, called the RESET operation, we can RESET them together to avoid interference.) We call this model the *crystalline-domain model*, because the domains have a different state from the cell base when they are crystalline. The *amorphous-domain model*, where the cell base is crystalline and the domains can be amorphous, can also be defined. Due to the space limitation, we omit its details, and focus on the crystalline-domain model.

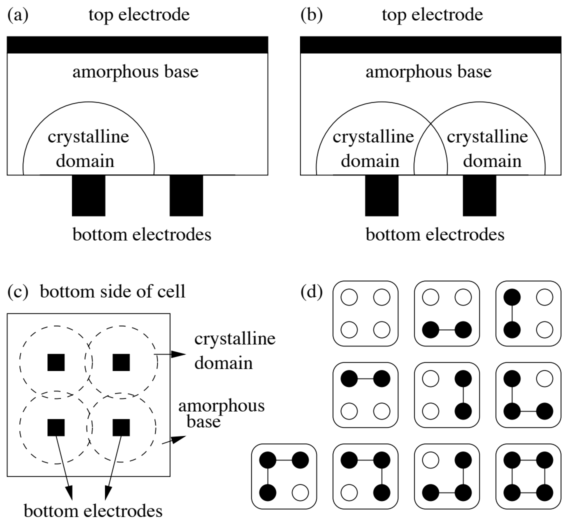

Fig. 1. Patterned cell with the crystalline-domain model. (a) A PCM cell with two bottom electrodes and one crystalline domain. The two bottom electrodes are not connected (i.e., there is high resistance between them). (b) The two bottom electrodes are connected by two overlapping crystalline domains. (c) The bottom-side view of a cell with $n = 4$ potential crystalline domains. (d) The 10 different connectivity patterns for the 2×2 rectangular array of domains shown in (c). The black vertices are crystalline domains (called "on" vertices); the white vertices are not crystalline (called "off" vertices). The edges between vertices denote their connectivity.

We let every domain have two basic states: *on* (crystalline) or *off* (amorphous). If two neighboring domains are both on, they overlap and become electrically connected (i.e., low resistance). The connectivity of domains can be detected by measuring the resistance between their bottom electrodes, which uses low reading voltage and does not change the state of the domains. We use the connectivity patterns of domains to represent data. As an example, the connectivity patterns of the four domains in Fig. 1 (c) are illustrated in Fig. 1 (d).

Patterned cell is a new approach to store data using the internal structure of domains in PCM cells. The two basic states of its domains may eliminate the high precision and power requirements imposed by programming cell levels. The data representation scheme is a new type of code defined by graph connectivity. In this paper, we explore this new scheme, analyze its storage capacity, and study its error-correction capability and the construction of error-control codes.

Due to space limitation, we present the proofs of a number of theorems and code constructions in [3].

II. Storage Capacity of Patterned Cell

In this section, we present the graph model for connectivity-based data representation. Then we analyze the storage capacity of domains that form one or two dimensional arrays.

A. Graph Model for Connectivity-based Data Representation

Let $G = (V, E)$ be a connected undirected graph, whose vertices V represent the domains in a cell. An edge (u, v) exists if the two domains are adjacent (which means they overlap if they are both *on*). Let $S : V \to \{0, 1\}$ denote the states of vertices: $\forall\ v \in V$, $S(v) = 1$ if v is *on*, and $S(v) = 0$ if v is *off*. Denote the $|V|$ vertices by $v_1, v_2, \cdots, v_{|V|}$. We call $\left(S(v_1), S(v_2), \cdots, S(v_{|V|})\right)$ a *configuration* of G. Let $\tilde{U} = \{0, 1\}^{|V|}$ denote the set of all configurations. Since in the crystalline-domain model, the purpose of making a domain crystalline is to connect it to at least one crystalline neighbor, we focus on configurations denoted by $\mathcal{U}$ that satisfy this property: "For any $v \in V$ that is on, at least one of its neighbors is also on." That is, $\mathcal{U} = \{\left(S(v_1), S(v_2), \cdots, S(v_{|V|})\right) \in \tilde{U} \mid \forall 1 \le i \le |V|$, if $S(v_i) = 1$, then $\exists v_j \in V$ such that $(v_i, v_j) \in E$ and $S(v_j) = 1\}$. We call $\mathcal{U}$ the set of *valid* configurations.

Let $C : V \times V \to \{0, 1\}$ denote the connectivity between vertices: "$\forall\ w_1 \neq w_2 \in V$, $C(w_1, w_2) = 1$ if there exists a sequence of vertices $(w_1 = u_1, u_2, \cdots, u_k = w_2)$ such that $(u_i, u_{i+1}) \in E$ and $S(u_i) = S(u_{i+1}) = 1$ for $i = 1, 2, \cdots, k-1$; otherwise, $C(w_1, w_1) = 0$. And for any $w \in V$, we set $C(w, w) = 1$ by default." Two vertices w_1, w_2 are *connected* if $C(w_1, w_2) = 1$. The vector $(C(v_1, v_1), C(v_1, v_2), \cdots, C(v_1, v_{|V|}); C(v_2, v_1), C(v_2, v_2), \cdots, C(v_2, v_{|V|}); \cdots\cdots; C(v_{|V|}, v_1), C(v_{|V|}, v_2), \cdots, C(v_{|V|}, v_{|V|}))$ is called the *connectivity pattern* of G. Clearly, not all vectors in $\{0, 1\}^{|V| \times |V|}$ are connectivity patterns that correspond to valid configurations (or even just configurations). So to be specific, let $f : \mathcal{U} \to \{0, 1\}^{|V| \times |V|}$ be the function that maps a valid configuration to its connectivity pattern. Let $\mathcal{C} = \{f(\vec{u}) \mid \vec{u} \in \mathcal{U}\}$, and we call $\mathcal{C}$ the set of *valid connectivity patterns.*

Lemma 1. *The mapping $f : \mathcal{U} \to \mathcal{C}$ is a bijection.*

Proof: Given a connectivity pattern $\vec{c} \in \mathcal{C}$, we see that a vertex $v \in V$ is *on* if and only if it is connected to at least one neighbor. So the configuration is determined by $\vec{c}$. ■

A PCM can read the connectivity pattern. We store data by mapping elements in $\mathcal{C}$ to symbols. The rate of graph G is $\frac{\log_2 |\mathcal{C}|}{|V|} = \frac{\log_2 |\mathcal{U}|}{|V|}$ bits per vertex (i.e., domain).

B. Capacity of One-dimensional Array

It is not difficult to compute the rate of G when $|V|$ is small. In this paper, we focus on large $|V|$ (especially for $|V| \to \infty$), which corresponds to using numerous domains in a large PCM layer. Let $n = |V|$, and define $N(n) \triangleq |\mathcal{C}| = |\mathcal{U}|$. We define the *capacity* of G as $cap = \lim_{n\to\infty} \frac{log_2 N(n)}{n}$.

We first consider the case where the domains form a one-dimensional array. That is, in graph $G = (V, E)$, we have $V = \{v_1, v_2, \cdots, v_n\}$ and $E = \{(v_1, v_2), (v_2, v_3), \cdots, (v_{n-1}, v_n)\}$. We denote the *capacity* of the one-dimensional array by cap_{1D}.

Theorem 2. *Let* $\lambda^* = \frac{1}{6}\left(100 + 12 \times \sqrt{69}\right)^{1/3} + \frac{2}{3\left(100+12\times\sqrt{69}\right)^{1/3}} + \frac{2}{3} \approx 1.7549$. *We have*

$$cap_{1D} = \log_2 \lambda^* \approx 0.8114.$$

Proof: The valid configuration of a one-dimensional array is a constrained system, where every run of 1s (i.e., "on" vertices) needs to have length at least two. The Shannon cover of the system is shown in Fig. 2. Its adjacency matrix is $A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$. By solving $|A - \lambda I| = -(\lambda^3 - 2\lambda^2 + \lambda - 1) = 0$, we find that for matrix A, its eigenvalue of the greatest absolute value is $\lambda^* \approx 1.7549$. It is known that the capacity of the constrained system is $\log_2 \lambda^*$. ■

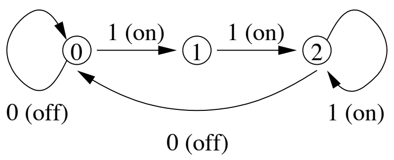

Fig. 2. Shannon cover for one-dimensional array.

We further present the number of valid configurations for a one-dimensional array with n vertices.

Theorem 3. *Let $\alpha_1, \alpha_2, \alpha_3$ be the three solutions to x for the equation $x^3 - 2x^2 + x - 1 = 0$, and let μ_1, μ_2, μ_3 be the numbers that satisfy the linear equation set*

$$\begin{cases} \mu_1\alpha_1 + \mu_2\alpha_2 + \mu_3\alpha_3 = 1 \\ \mu_1\alpha_1^2 + \mu_2\alpha_2^2 + \mu_3\alpha_3^2 = 2 \\ \mu_1\alpha_1^3 + \mu_2\alpha_2^3 + \mu_3\alpha_3^3 = 4 \end{cases}$$

(We get $\alpha_1 = \frac{1}{6} \cdot (100 + 12\sqrt{69})^{\frac{1}{3}} + \frac{2}{3} \cdot (100 + 12\sqrt{69})^{-\frac{1}{3}} + \frac{2}{3} \approx 1.7549$, $\alpha_2 = -\frac{1}{12} \cdot (100 + 12\sqrt{69})^{\frac{1}{3}} - \frac{1}{3} \cdot (100 + 12\sqrt{69})^{-\frac{1}{3}} + \frac{2}{3} + i \cdot (\frac{\sqrt{3}}{12} \cdot (100 + 12\sqrt{69})^{\frac{1}{3}} - \frac{\sqrt{3}}{3} \cdot (100 + 12\sqrt{69})^{-\frac{1}{3}}) \approx 0.1226 + 0.7449i$, $\alpha_3 = -\frac{1}{12} \cdot (100 + 12\sqrt{69})^{\frac{1}{3}} - \frac{1}{3} \cdot (100 + 12\sqrt{69})^{-\frac{1}{3}} + \frac{2}{3} - i \cdot (\frac{\sqrt{3}}{12} \cdot (100 + 12\sqrt{69})^{\frac{1}{3}} - \frac{\sqrt{3}}{3} \cdot (100 + 12\sqrt{69})^{-\frac{1}{3}}) \approx 0.1226 - 0.7449i$, $\mu_1 \approx 0.7221$, $\mu_2 \approx 0.1389 + 0.2023i$, *and* $\mu_3 \approx 0.1389 - 0.2023i$.*) Then for a one-dimensional array with n vertices, we have* $N(n) = |\mathcal{C}| = |\mathcal{U}| = \mu_1\alpha_1^n + \mu_2\alpha_2^n + \mu_3\alpha_3^n$.

	Lower (Tiling)	Lower (Bit-Stuffing)	Upper Bound
Rectangular	0.959338	0.961196	0.963109
Triangular	0.987829	0.987218	0.990029

TABLE I
UPPER AND LOWER BOUNDS FOR TWO-DIMENSIONAL ARRAY'S CAPACITY.

C. Capacity of Two-dimensional Arrays

We now consider the case where the domains form a two-dimensional array. Specifically, we study two types: the *rectangular array* and the *triangular array*, illustrated in Fig. 3. We denote the capacity of the two-dimensional array by *cap*. Some existing techniques based on convex/concave programming, including tiling, bit-stuffing, *etc.*, can be applied here to obtain the upper and lower bounds of the capacity. We summarize the bounds in Table I. It is interesting that the capacity is really high (close to 1) for both arrays. In the rest of this section, we will discuss the bounds in detail.

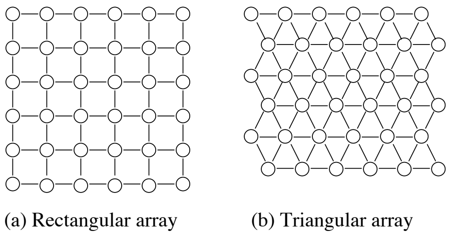

Fig. 3. Two types of two-dimensional arrays.

1) Lower Bound based on Tiling: If we consider a distribution θ on the valid configuration set $\mathcal{U}$, then the rate of G is $R(\theta) = \frac{H(\theta)}{n}$. So another expression for capacity is $cap = \max_\theta \lim_{n\to\infty} R(\theta)$. For any distribution θ, $\lim_{n\to\infty} R(\theta)$ is a lower bound for *cap*. Different ways of constructing θ lead us to different methods.

In [5], Tiling was proposed as a variable-length encoding technique for two-dimensional (2-D) constraints, such as runlength-limited (RLL) constraints and no isolated bits (n.i.b.) constraints. The idea of tiling is that we can divide all the 2-D plane using shifted copies of two certain shapes, referred as 'W' and 'B' tiles. Here, we say that a set of vertices A is a shift or shifted copy of another set B if and only if their vertices are one-to-one mapped and the position movement (vector) between each vertex in A and its corresponding vertex in B is fixed. For these two types of tiles – 'W' tiles and 'B' tiles, – they have the following properties:

1) The 'W' tiles are freely configurable. That means given any configuration for all the 'W' tiles, we can always find a configuration for all the 'B' tiles such that they satisfy the 2-D constraints.
2) Given any configuration for all the 'W' tiles, the configurations for the 'B' tiles are independent with each other.

According to these properties, we can first set 'W' tiles independently based on a predetermined distribution π, and then configure the 'B' tiles uniformly and independently (given the 'W' tiles). Finally, the maximal information rate $\max_\pi R(\pi)$ is a lower bound of the array's capacity.

Note that the constraint for valid configurations is that each "on" vertex has at least one "on" neighbor. For rectangular/triangular arrays, we can use tiling schemes in Fig. 4.

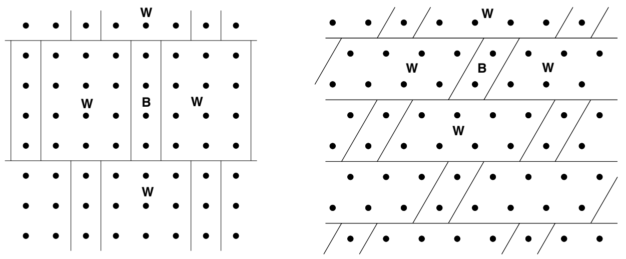

Fig. 4. Tiling schemes for the rectangular (left) and triangular (right) arrays.

According to Theorem 3.1 in [5], we have

$$cap \geq \max_\pi R(\pi) = \max_\pi \frac{H(\pi) + \sum_\phi P_\pi(\phi)|S(\phi)|}{|W| + |B|}.$$

Here, $|W|$ (or $|B|$) is the size of each 'W' ('B') tile, e.g., $|W| = 12$ in the left-side tiling of Fig. 4 and $|B| = 2$ in the right-side tiling of Fig. 4; $H(\pi)$ is the entropy corresponding to distribution π; ϕ is the configuration of the 'W' blocks around a 'B' block (four blocks in Fig. 4), whose distribution is a function of π, denoted as $P_\pi(\phi)$; $|S(\phi)|$ is the number of available distinct configurations for a 'B' blocks given the 'W' blocks around it. Based on this formula, we are able to get the lower bounds in the first column of Table I using convex programming with linear constraints.

2) Lower Bound based on Bit-Stuffing: Another way to obtain the lower bounds for the capacities of 2-D constraint codes is based on bit-stuffing [6]. In bit-stuffing, let ∂ denote the vertices near the left and top boundaries, called boundary vertices. Assume we know the state configuration of ∂; then we can program the remaining vertices one by one such that the ith vertex depends on a set of programmed vertices near it, denoted by D_i. In this scheme, for different i, j, we have that the set $D_i \bigcup i$ is a shift of the set $D_j \bigcup j$, and for all i, the conditional distribution $P(x_i|x(D_i))$ is fixed, denoted by γ, where $x(D_i)$ is the configuration of D_i.

Let θ denote the probability distribution of the configuration on all the vertices V, and let δ denote the probability distribution of the configuration on the boundary islands ∂. Then we see that θ is uniquely determined by δ and the conditional distribution γ. It is not hard to prove that for any conditional distribution γ, when the 2-D array is infinitely large, there exists a distribution δ such that θ is stationary. That means for any subset $A \subset V$ and its arbitrary shift $\sigma(A) \subset V$, A and $\sigma(A)$ have the same configuration distribution, namely,

$$P_\theta(x(A) = a) = P_\theta(x(\sigma(A)) = a)$$

for any state configuration a. Note that this equation is true only when the block is infinity large; otherwise, θ is quasi-stationary [6].

Given this stationary distribution θ, we would like to calculate the relative entropy R_i of the ith vertex given the states of the vertices programmed before it. (Here the ith vertex is not a boundary vertex). Assume the state distribution on D_i is ϕ; then according to the definition of bit-stuffing

$$R_i = \sum_{y\in\{0,1\},z\in\{0,1\}^{|D_i|}} \phi(z)H(\gamma(y|z))$$

where $|D_i|$ is the same for different i, so we can also write it as $|D|$. It is not easy to get the exact value of R_i because ϕ is unknown (it depends on γ) and there are too many constraints to guarantee that θ is stationary. By relaxing the constraints, we get a set of distributions on D_i, denoted as $\{\phi'\}$, such that θ is stationary near the ith vertex (limited in a fixed area T near the ith vertex). Therefore,

$$R_i \geq \min_{\phi'} \sum_{y\in\{0,1\},z\in\{0,1\}^{|D|}} \phi'(z)H(\gamma(y|z))$$

such that (1) the configuration distribution on T is stationary, and (2) given some $z \in \{0,1\}^{|D|}$, we have $\gamma(0|z) = 0$ to guarantee that each "on" vertex has at least one "on" neighbor.

Since the inequality above holds for all the vertices except the boundary vertices, a lower bound of the capacity can be written as

$$\max_{\gamma}\min_{\phi'}\sum_{z}\phi'(z)H(\gamma(y|z))$$

under the constraints. (For more discussions, please see [6].)

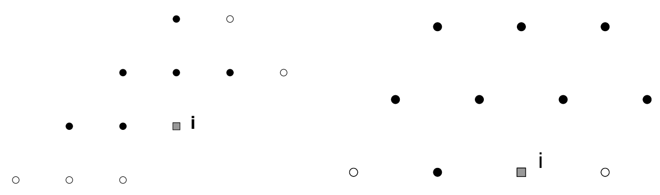

Fig. 5. The bit-stuffing schemes for the rectangular and triangular arrays.

Fig. 5 shows the bit-stuffing schemes that we use to calculate the lower bounds of the 2-D arrays' capacities. In this figure, the vertex i is marked as a gray square; D_i is indicated by the black vertices that the vertex i depends on; the stationary constraint is applied to the region T that includes all the vertices plotted. Based on these schemes, we get the lower bounds for the capacities, which are given in the second column in Table I.

3) Upper Bound based on Convex Programming: In [7], convex programming was used as a method for calculating an upper bound on the capacity of 2-D constraints. The idea is based on the observations that there exists an optimal distribution θ^* such that θ^* is stationary and symmetric when the array is sufficiently large. The stationary property implies that for any set of vertices A, – let $\sigma(A)$ be an arbitrary shift of A, – A and $\sigma(A)$ have the same state (configuration) distribution. The symmetric property depends on the type of the array. For a rectangular array, if two sets of vertices A and B are reflection symmetric about a horizontal/vertical line or a 45-degree line, then they have the same state (configuration) distribution. Note that the reflection symmetry about a 45-degree line is also called transposition invariance in [7]. For a triangular array, there are more symmetries: if two sets of vertices A and B are reflection symmetric about a horizontal/vertical line or a 30/60-degree line, then they have the same state (configuration) distribution.

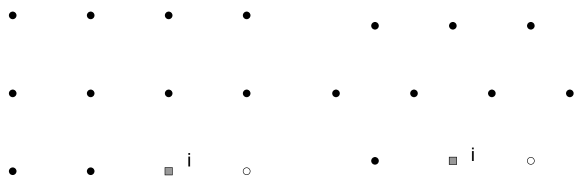

Fig. 6. The schemes for calculating the upper bounds of the capacities.

Now let us consider the distribution over a small region T for both arrays, as shown in Fig. 6. For example, in the rectangular array, assume the distribution on T (the 12 vertices) is ϕ; then given the first ten vertices, the relative entropy of the next vertex is a function of ϕ, denoted by $R(\phi)$. Let's index all the vertices by $1, 2, 3, ..., n$ from left to right and then from top to bottom and let $R_i = H(x_i|x_1, x_2, ..., x_{i-1})$. It is easy to see that if a vertex i is not on the boundary, then

$$R_i \leq H(x_i|\{x_1, x_2, ..., x_{i-1}\}\bigcap T) = R(\phi).$$

That implies that $R(\phi)$ is an upper bound for

$$cap = \lim_{n\to\infty}\max_{\theta}\frac{\sum_{i=1}^{n} R_i}{n}$$

So our work is to maximize $R(\phi)$ such that ϕ is stationary and symmetric on T. Thus we get the upper bounds for the capacity of the rectangular array in Table I. The same method also applies to the triangular array.

III. Error Correction and Detection

In this section, we study error correction/detection for patterned cells. We focus on one-dimensional arrays and two-dimensional rectangular arrays. When programming domains, a potentially important type of error is to make a domain too large such that it changes the connectivity pattern unintentionally. Two kinds of such errors are shown in Fig. 7, where in (a) two diagonal "on" domains overlap, and in (b) an "on" domain touches its neighboring "off" domain's bottom electrode. It can be proved that the former kind of errors can always be corrected, because the two concerned domains' states can be correctly determined by checking if they are connected to one of their four neighbors. So in this paper, we focus on the latter kind of error, which is important and less trivial. We call the latter error an *overreach error*, which happens only between an "on" vertex and a neighboring "off" vertex, and the error makes them become connected. We assume that between every pair of neighboring "on" and "off" vertices, the overreach error happens independently with probability p_e. Given p_e, we define the *capacity* as the maximum number of bits that can be stored per vertex such that the data can be decoded correctly with high probability (which approaches one as the array's size approaches infinity).

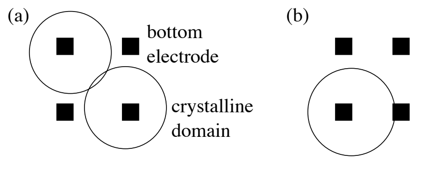

Fig. 7. Error models. (a) Two diagonal domains overlap. (b) Overreach error.

A. One-dimensional Array

Let $G = (V, E)$ be a one-dimensional array of n vertices: $v_1, v_2, \cdots, v_n$. When $n \to \infty$ and given the overreach error probability p_e, let $cap_1(p_e)$ denote its capacity.

Theorem 4. *For one-dimensional array,* $cap_1(p_e) \geq$

$$\max\{0.5, \max_{x\in[0,0.4]} x(1 - H(p_e)) + \frac{2-x}{4} H\left(\frac{4x}{2-x}\right)\}.$$

Proof: We present the sketch of the proof here. For details of the proof, please see [3]. To show $cap_1(p_e) \geq 0.5$, partition the n cells in the array into pairs, where every pair is either both *on* or both *off* and stores one bit; the code can correct all overreach errors. To show $cap_1(p_e) \geq \max_{x\in[0,0.4]} x(1 - H(p_e)) + \frac{2-x}{4} H\left(\frac{4x}{2-x}\right)$, consider the following code.

Let $m \in \mathbb{Z}^+$ be even. Given a binary vector $\vec{d} = (d_1, d_2, \cdots, d_m)$, define $\Delta(\vec{d})$ as $\Delta(\vec{d}) = (d_1, d_2 + d_1 \mod 2, d_3 + d_2 \mod 2, \cdots, d_m + d_{m-1} \mod 2)$. It can be shown that when $m \to \infty$, there exists a binary code $\mathcal{D}$ of rate $1 - H(p_e)$ that can correct binary symmetric errors of error probability p_e, such that for every codeword $\vec{d} \in \mathcal{D}$, the Hamming weight of $\Delta(\vec{d})$ equals $m/2$.

Let $n \geq \frac{5}{2}m+2$, and let $n - \frac{m}{2}$ be even. Let $\mathcal{C} \subset \{0,1\}^n$ be a code for the one-dimensional array of n vertices, where every codeword $\vec{s} = (s_1, s_2, \cdots, s_n) \in \mathcal{C}$ is a valid configuration that satisfies these conditions:

1) The vector $\vec{s}$ has $m+1$ 1-runs and 0-runs, where every 1-run or 0-run has at least two vertices.
2) Let $L_1, L_2, \cdots, L_{m+1}$ denote the run-lengths of the $m+1$ 1-runs and 0-runs in $\vec{s}$. Define the *signature* of $\vec{s}$ as $sig(\vec{s}) = (L_1 \mod 2, \sum_{i=1}^{2} L_i \mod 2, \sum_{i=1}^{3} L_i \mod 2, \cdots, \sum_{i=1}^{m} L_i \mod 2)$. Then $sig(\vec{s}) \in \mathcal{D}$.

It can be shown that the code $\mathcal{C}$ can tolerate overreach errors of error probability p_e, by transforming the overreach errors in a codeword $\vec{s} \in \mathcal{C}$ to binary symmetric errors in its signature $sig(\vec{s}) \in \mathcal{D}$. Furthermore, every signature in $\mathcal{D}$ corresponds to $2\binom{\frac{n}{2}-\frac{m}{4}-1}{m}$ codewords in $\mathcal{C}$, which gives us the rate of code $\mathcal{C}$. By letting $x = m/n$, we get the conclusion. ■

It is noticeable that the overreach error is a type of asymmetric error for graph connectivity. We have constructed an error-detecting code that can detect *all* overreach errors. Its underlying idea is closely related to the well-known Berger code [1] for asymmetric errors. Due to space limitation, we leave the detailed code construction in [3]. The code leads to the following theorem.

Theorem 5. *Let $m \geq 2$ be an integer. Let r be the smallest positive integer such that $\mu_1\alpha_1^r + \mu_2\alpha_2^r + \mu_3\alpha_3^r \geq m$. (The constants $\alpha_1, \alpha_2, \alpha_3, \mu_1, \mu_2, \mu_3$ are specified in Theorem 3.) Then, there is an error-detecting code of length $m + r$ and rate*

$$\frac{\log_2(\mu_1\alpha_1^m + \mu_2\alpha_2^m + \mu_3\alpha_3^m)}{m + r}$$

bits per vertex that can detect all overreach errors. When $m \to \infty$, we have $r = \log_{\alpha_1} m \approx \log_{1.7549} m$, and the rate of the code is $cap_{1D} = \log_2 \alpha_1 \approx 0.8114$, which reaches the capacity of one-dimensional arrays.

B. Two-dimensional Array

We now focus on the capacity of two-dimensional rectangular array when i.i.d. overreach errors happen with probability p_e between neighboring *on* and *off* vertices. Let $G = (V, E)$ be an $m \times m$ two-dimensional rectangular array, where $m \to \infty$. Let $cap_2(p_e)$ denote its capacity. It can be seen that as $p_e \to 0$, the lower bound in the next theorem approaches $4/5$.

Theorem 6. *For any $q \in [0, 1/2]$, let $\eta(q, p_e) = (1 - q^3)(p_e + (1 - p_e)(1 - (1 - (1 - q)p_e)^3))$. Then for two-dimensional rectangular array,*

$$cap_2(p_e) \geq \frac{4}{5} \max_{q\in[0,0.5]} H(1 - q + q\eta(q, p_e)) - qH(\eta(q, p_e)).$$

ACKNOWLEDGMENT

This work was supported in part by the NSF CAREER Award CCF-0747415, the NSF grant ECCS-0802107, and by an NSF-NRI award.

REFERENCES

[1] J. M. Berger, "A note on an error detection code for asymmetric channels," *Information and Control*, vol. 4, pp. 68–73, March 1961.
[2] G. W. Burr *et al.*, "Phase change memory technology," *Journal of Vacuum Science and Technology*, vol. 28, no. 2, pp. 223–262, March 2010.
[3] A. Jiang, H. Zhou, Z. Wang and J. Bruck, "Patterned cells for phase change memories," Caltech Technical Report, 2011. Online: $http://www.paradise.caltech.edu/etr.html$.
[4] D. Lammers, "Resistive RAM gains ground," in *IEEE Spectrum*, pp. 14, September 2010.
[5] A. Sharov and R. M. Roth, "Two-Dimensional constrained coding based on tiling", *IEEE Trans. Inform. Th.*, vol. 56, no. 4, pp. 1800–1807, 2010.
[6] I. Tal and R. M. Roth, "Bounds on the rate of 2-D bit-stuffing encoders", *IEEE Trans. on Information Theory*, vol. 56, no. 6, pp 2561-2567, 2010.
[7] I. Tal and R. M. Roth, "Convex programming upper bounds on the capacity of 2-D constraints", *IEEE Transactions on Information Theory*, vol. 57, no. 1, pp 381–391, 2011.