Error-Correcting Schemes with Dynamic Thresholds in Nonvolatile Memories

Hongchao Zhou
Electrical Engineering Department
California Institute of Technology
Pasadena, CA 91125
Email: hzhou@caltech.edu

Anxiao (Andrew) Jiang
Computer Science and Engineering Department
Texas A&M University
College Station, TX 77843
Email: ajiang@cse.tamu.edu

Jehoshua Bruck
Electrical Engineering Department
California Institute of Technology
Pasadena, CA 91125
Email: bruck@caltech.edu

***Abstract*—Predetermined fixed thresholds are commonly used in nonvolatile memories for reading binary sequences, but they usually result in significant asymmetric errors after a long duration, due to voltage or resistance drift. This motivates us to construct error-correcting schemes with dynamic reading thresholds, so that the asymmetric component of errors are minimized. In this paper, we discuss how to select dynamic reading thresholds without knowing cell level distributions, and present several error-correcting schemes. Analysis based on Gaussian noise models reveals that bit error probabilities can be significantly reduced by using dynamic thresholds instead of fixed thresholds, hence leading to a higher information rate.**

I. Introduction

Nonvolatile memories, like EPROM, EEPROM, Flash memory or Phase-change memory (PCM), are memories that can keep the data content even without power supply. This property enables them to be used in a wide range of applications, including cell-phones, consumers, automotive and computers. Many research studies have been carried out on nonvolatile memories because of their unique features, attractive applications and huge marketing demands.

An important challenge for most nonvolatile memories is data reliability. The stored data can be lost due to many mechanisms, including cell heterogeneity, programming noise, write disturbance, read disturbance, etc [1], [2]. From a long-term view, the change in data has an asymmetric property. For example, the stored data in flash memories is represented by the voltage levels of transistors, which drift in one direction because of charge leakage. In PCM, another class of nonvolatile memories, the stored data is determined by the electrical resistance of the cells, which also drifts due to thermally activated crystallization of the amorphous material. All these mechanisms make the errors in nonvolatile memories heterogeneous, asymmetric, time-dependent and unpredictable. That brings substantial difficulties to researchers attempting to develop simple and efficient error-correcting schemes.

To date, existing coding schemes for nonvolatile memories commonly use fixed thresholds to read data. For instance, in flash memories, a threshold voltage level v_{th} is predetermined such that if the voltage level of a given cell is higher than v_{th} it reads '1', and otherwise it reads '0'. To increase the data reliability, error-correcting codes including Hamming code, BCH code, Reed-Solomon code and LDPC code are applied in nonvolatile memories to combat errors. Because of the asymmetric feature of nonvolatile memories, given a fixed threshold, the probability of $1 \to 0$ errors are usually much higher than $0 \to 1$ errors after a long duration. Consequently, the concept of asymmetric error-correcting codes was proposed and investigated in the literature [3]–[5], in which only $1 \to 0$ errors are allowed. However, in practice, the errors in nonvolatile memories are neither purely asymmetric errors nor purely symmetric errors, which limits the applications of asymmetric error-correcting codes in nonvolatile memories.

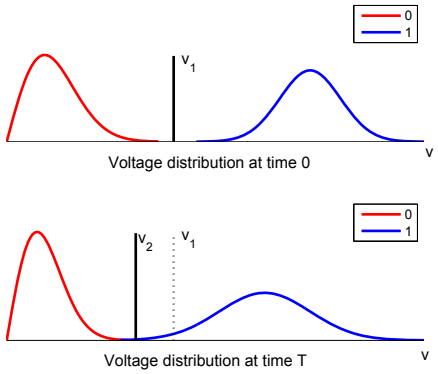

Fig. 1. An illustration of the voltage distributions for bit '1' and bit '0' in flash memories. The top figure is for newly written data, and the bottom figure is for old data that has been stored for a long time T.

To overcome the limitations of fixed thresholds in reading data in nonvolatile memories, dynamic thresholds are introduced in this paper. To better understand this, we use flash memories for illustration, see Fig. 1. In the figure, assume the left curve indicates the voltage distribution for bit '0' (a bit '0' is written during programming) and the right curve indicates the voltage distribution for bit '1'. At time 0 (the moment after programming), it is best to set the threshold voltage as $v_{th} = v_1$, for separating bit '1' and '0'. But after a period of time, the voltage distribution will change. In this case, v_1 is no longer the best choice, since it will introduce too many $1 \to 0$ errors. Instead, we can set the threshold voltage as $v_{th} = v_2$ (see the second plot in the figure), to minimize the error probability. This also applies to other nonvolatile memories, such as PCMs. In view of these considerations, the scope of

978-1-4577-0594-6/11/$26.00 ©2011 IEEE

this paper lies in designing efficient error-correcting schemes with dynamic reading thresholds in nonvolatile memories.

For convenience, we consider different types of nonvolatile memories in the same framework where the data is represented by cell levels, such as voltages in flash memories and resistance in PCMs. Let n denote the number of cells in a block, and $v_1, v_2, ..., v_n \in \mathcal{R}^n$ be the levels of these n cells. Let t indicate the time since the data has been written. The cell levels may be disturbed or affected by many mechanisms, therefore they are time-dependent.

One important problem is to determine the threshold value for a given block with n cells at time t. If the level distributions for bit '1' and '0' at any time t are accurately provided, the problem can be easily solved by minimizing the expected error probability. Unfortunately, this task is infeasible in practice because of the lack of time records, the heterogeneity of blocks, and the unpredictability of exceptions. Another possible method is to apply unsupervised clustering to all the cell levels so that they are classified into two groups corresponding to bit '1' and bit '0'. But in practice, the border between bit '1' and '0' may become more and more fuzzy, and mistakes of clustering may cause significant reading errors. To determine the threshold dynamically, our idea is to adjust the threshold such that there are always k cells with higher levels than this threshold in a block. Based on this idea, two error-correcting schemes are proposed for nonvolatile memories. The first scheme is based on Berger codes. In this scheme, k is different for different blocks, hence, it is stored as metadata. When we read data from a block, we first read the value of k based on a fixed threshold, and then read the codeword from the block with an adjusted threshold such that there are k ones in the block. The second scheme is based on balanced codes, where $k = n/2$, i.e., the number of ones equals to the number of zeros in each block. Hence, it does not have to store k for adjusting the reading threshold.

The rest of the paper is organized as follows. In Section II, we analyze the performance of dynamic thresholds, and compare it with some other thresholds such as fixed thresholds. Two error-correcting schemes with dynamic thresholds, based on Berger codes and balanced codes respectively, are proposed and discussed in Section III and IV. Section V studies how to calculate dynamic thresholds fast, followed by the conclusion.

II. Performance Analysis of Dynamic Thresholds

In a block of length n, let $x = (x_1, x_2, ..., x_n) \in \{0,1\}^n$ be the word written in this block and let $v_1, v_2, ..., v_n$ be the cell levels for reading. The number of 1's written in the block is $k = \sum_i x_i$. Given any threshold level v_{th}, we read a word $y(v_{th}) = (y_1, y_2, ..., y_n)$ such that $y_i = 1$ if and only if $v_i \geq v_{th}$. As a result, the number of errors based on v_{th} is $N_e(v_{th}) = |y(v_{th}) - x|$, where $|y - x|$ is the Hamming distance between two binary sequences x and y.

A dynamic threshold, denoted as v_d, is selected such that there are exactly k cells in the block will be read as ones, i.e., $|y(v_d)| = k = |x|$. In order to recover x from $y(v_d)$ reliably, it requires error-correcting codes, which will be discussed later. In this section, we mainly focus on the performance analysis of dynamic thresholds.

A. Sub-Optimality of Dynamic Thresholds

Let's first compare dynamic thresholds with optimal thresholds. We say that one threshold $v_{th} = v^*$ is optimal if and only if $N_e(v_{th})$ is minimized at v^*, i.e., $v^* = \arg\min_{v_{th}} N_e(v_{th})$. However, v^* is imaginary, since we cannot determine it without knowing the initial word x. The following theorem shows that dynamic thresholds have performance comparable to optimal thresholds. Even in the worst case, the number of errors introduced based on v_d is at most double of that introduced by v^*.

Theorem 1. *For any cell levels $v_1, v_2, ..., v_n$, we have*

$$N_e(v_d) \leq 2N_e(v^*)$$

Proof: Given the threshold v_d, the number of $0 \to 1$ errors equals the number of $1 \to 0$, denoted by $N_e^+(v_d) = N_e^-(v_d)$. Hence,

$$N_e(v_d) = 2N_e^+(v_d) = 2N_e^-(v_d)$$

If $v^* \geq v_d$, the number of $1 \to 0$ errors $N_e^-(v^*) \geq N_e^-(v_d)$. Therefore,

$$N_e(v_d) \leq 2N_e^-(v^*) \leq 2N_e(v^*)$$

Similarly, if $v^* < v_d$, by considering $0 \to 1$ errors, we get the same conclusion. ■

If we consider the worst case for a fixed threshold, denoted as v_f, it may introduce as many as n errors while v^* or v_d introduces zero errors. For an instance, we assume $x = 111..11$ and we set $v_f = \max_{i=1}^n v_i + \epsilon$ with $\epsilon > 0$. In this case, the number of errors is n. One solution for v^* is $v^* = \min_{i=1}^n v_i - \epsilon$ with $\epsilon > 0$. In this case, the number of errors based on v^* or v_d is 0. We see that for the worst case that we completely don't know the distributions of cell levels, dynamic thresholds have sub-optimal performance, which is much better than that of fixed thresholds.

B. A Statistical View

To better understand the different types of thresholds as well as their performances, we study them from the expectation (statistical) perspective. Assume that we write n bits (including k ones) into a block at time 0, let $g_t(v)$ denote the p.d.f of the cell level for a bit 0 at time t, and let $h_t(v)$ denote the p.d.f of the cell level for a bit 1 at time t. Then as n becomes sufficiently large, based on a dynamic threshold v_d, we have

$$N_e^+(v_d) \to E(N_e^+(v_d)), \quad N_e^-(v_d) \to E(N_e^-(v_d))$$

where N_e^+ indicates the number of $0 \to 1$ errors and N_e^- indicates the number of $1 \to 0$ errors. According to the definition of dynamic thresholds, we have $N_e^+(v_d) = N_e^-(v_d)$. This implies that $E(N_e^+(v_d)) = E(N_e^-(v_d))$ approximately when n is large, i.e.,

$$(n-k)\int_{v=v_d}^{\infty} g_t(v)dv = k\int_{v=-\infty}^{v_d} h_t(v)dv$$

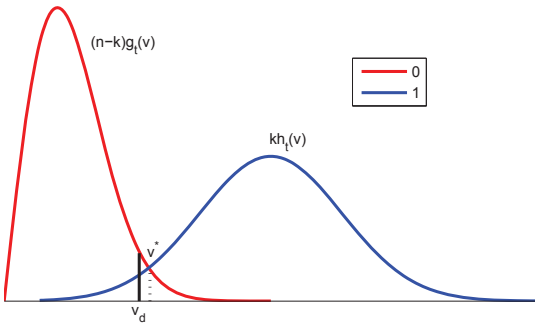

Fig. 2. An illustration of v_d and v^*.

As an illustration in Fig. 2, we see that the error regions on the left of v_d and on the right of v_d have almost the same area.

Differently, an optimal threshold v^* is chosen such that $N_e^+(v^*) + N_e^-(v^*)$ is minimized. Approximately, when n is large, we have

$$v^* = \arg\min_{v_{th}}(n-k)\int_{v=v_{th}}^{\infty} g_t(v)dv + k\int_{v=-\infty}^{v_{th}} h_t(v)dv$$

So when $g_t(v)$ and $h_t(v)$ are continuous functions, we have

$$v^* = \pm\infty \text{ or } (n-k)g_t(v^*) = kh_t(v^*)$$

That means v^* is approximately one of the intersection points of the two curves in Fig. 2 or one of the infinity points.

In the following examples, we assume that $g_t(v)$ and $h_t(v)$ are Gaussian distributed. For simplification, we assume that $k = n/2$, which is consistent with our schemes (will be presented later) in this paper. In this case, given a threshold v_{th}, the bit error rate of a block is

$$P_e(v_{th}) = \frac{1}{2}\int_{v_{th}}^{\infty} g_t(v)dv + \frac{1}{2}\int_{-\infty}^{v_{th}} h_t(v)dv$$

Example 1. *Let $g_t(v) = \mathcal{N}(0,\sigma)$ and $h_t(v) = \mathcal{N}(1-t,\sigma)$. To compare different thresholds, we assume the fixed threshold $v_f = \frac{1}{2}$, which satisfies $g_0(v_f) = h_0(v_f)$.*

In the above example, the cell levels corresponding to bit '1' drift but their variance does not change. We have

$$v^* = v_d = \frac{1-t}{2}, \quad v_f = \frac{1}{2}$$

At time t, the bit error rate based on a threshold v_{th} is

$$P_e(v_{th}) = \frac{1}{2}\Phi(-\frac{v_{th}}{\sigma}) + \frac{1}{2}\Phi(-\frac{1-t-v_{th}}{\sigma})$$

where $\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-t^2/2}dt$.

For different thresholds, $P_e(v_{th})$ is plotted in Fig. 3, which shows the good performance of dynamic thresholds when cell levels drift.

Example 2. *Let $g_t(v) = \mathcal{N}(0,\sigma)$ and $h_t(v) = \mathcal{N}(1,\sigma+t)$. To compare different thresholds, we assume the fixed threshold $v_f = \frac{1}{2}$, which satisfies $g_0(v_f) = h_0(v_f)$.*

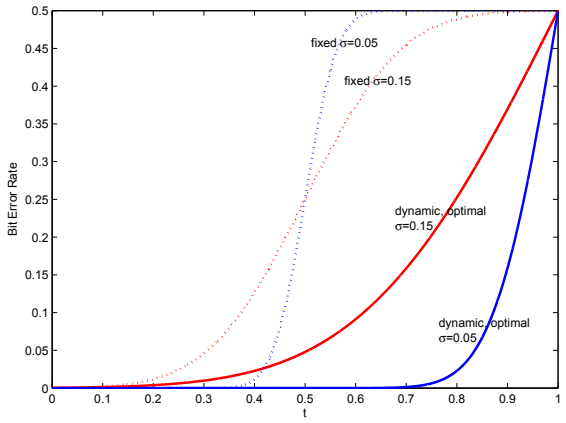

Fig. 3. Bit error rates as functions of time t, under the first model with $g_t(v) = \mathcal{N}(0,\sigma)$ and $h_t(v) = \mathcal{N}(1-t,\sigma)$.

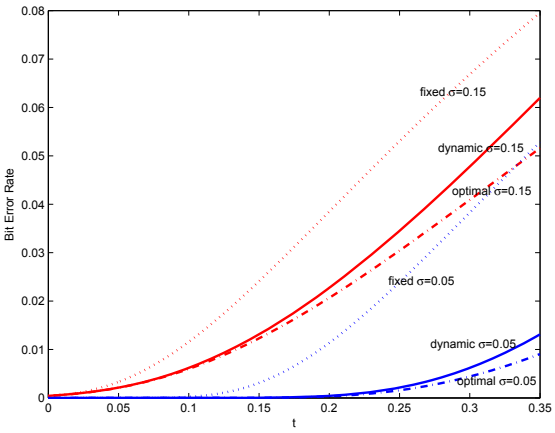

Fig. 4. Bit error rates as functions of time t, under the second model with $g_t(v) = \mathcal{N}(0,\sigma)$ and $h_t(v) = \mathcal{N}(1,\sigma+t)$.

In this example, the variance of the cell levels corresponding to bit '1' increases as t increases. We have

$$e^{-\frac{v^{*2}}{2\sigma^2}} = \frac{\sigma}{\sigma+t}e^{-\frac{(1-v^*)^2}{2(\sigma+t)^2}}, \quad v_d = \frac{1}{2+t/\sigma}, \quad v_f = \frac{1}{2}$$

At time t, the bit error rate based on a threshold v_{th} is

$$P_e(v_{th}) = \frac{1}{2}\Phi(-\frac{v_{th}}{\sigma}) + \frac{1}{2}\Phi(-\frac{1-v_{th}}{\sigma+t})$$

which is plotted in Fig. 4 for different thresholds. It shows that dynamic thresholds introduce less errors than fixed thresholds when bit '1' and '0' have different reliability (reflected by their variances).

In practice, the cell level distributions at a time t are much more complex than the Gaussian distributions, and the errors introduced are due to many mechanisms. However, the analysis based two simple models above are still useful, because they reflect the trend of the cell level changes, which is helpful for analyzing the time-dependent errors in nonvolatile memories.

III. Scheme Based on Berger Codes

In 1961, Berger designed a class of codes to detect all unidirectional errors in telecommunications [6], called Berger codes, where only $1 \to 0$ errors or only $0 \to 1$ errors are considered. Let u be the information word of n bits and let v the be the total number of ones in u. Then a codeword of u can be expressed as uv, where v is of length $r = \lceil \log_2(n+1) \rceil$ in binary representation. In this paper, we use the same format of Berger codes that v indicates the number of ones in u, but we use v as metadata for determining the dynamic thresholds, not for detecting errors. When we read a codeword $x = uv$, without considering errors, we can first read v based on a predetermined fixed threshold. Then we read the information word u based on an adjusted threshold corresponding to v. For practical use, u and v need to be protected against errors by using error-correcting codes.

In the scheme, two different thresholds are applied to read v and u separately, which results in different reliability. Correspondingly, we introduce two types of blocks, namely information blocks (for storing u) and metadata blocks (for storing v). They correspond to two error-correcting codes, denoted by C_1 and C_2, such that C_2 can correct more errors than C_1 because metadata blocks have a higher bit error rate due to fixed reading thresholds. Assume all the blocks have the same length n. Let k_1 and k_2 denote the dimensions of C_1 and C_2. Then $k_2 \leq k_1 < n$. For a given information word $u \in \{0,1\}^k$ with $k = k_1$, we can encode it in the following way: we first write the codeword $x = C_1(u)$ into an information block B_1, and then we write $v = w(x)$ into a metadata block B_2, where $w(x)$ is the number of ones in x. Usually, the length of v, i.e. $\lceil \log_2(n+1) \rceil$, is much smaller than k_2, which means that each metadata block can serve as many as $\lfloor \frac{k_2}{\lceil \log_2(n+1) \rceil} \rfloor$ information words.

In [5], the authors proposed flipping codes for correcting asymmetric errors. Their idea comes from the fact that the codes with higher Hamming weights are more prone to have errors in asymmetric channels, hence less reliable. It can be proved that, with dynamic thresholds, we are also able to reduce the bit error rates by flipping all the bits in a codeword if its weight is higher than $n/2$. Fortunately, this can be achieved in our scheme without introducing any additional bits. The only change is that given a codeword $x \in C_1$, if $w(x) > n/2$, we write the complement of x into an information block, denoted as $\overline{x}$, which is obtained by flipping all the bits in x. For decoding, we first check whether $v > \frac{n}{2}$. If $v > \frac{n}{2}$, by flipping all the bits again, the decoder gets a word $x + e$ with e as the error vector, which leads us to the original information word u. If $v \leq \frac{n}{2}$, the decoding process keeps unchanged.

The following example is constructed for the purpose of illustrating the procedure of encoding and decoding. Let C_1 be the $(7, 4)$ Hamming codes. When the information word is $u = 0110$, we have a codeword $x = 0110110$. Since $v = w(x) = 4 > n/2$, we write $\overline{x} = 1001001$ into an information block instead of x. For decoding, assume we can retrieve $v = 4$ correctly and the word read from the information block is $y = 1001101$, then applying the decoding algorithm of Hamming codes to the complement of y will bring us the information word u.

In the above example, the scheme is not efficient because the block length n is too small. In practice, n is usually much larger. Let's consider the following case: let $n = 255$ and let C_1 and C_2 be primitive BCH codes correcting 8 and 18 errors separately. In this case, we have $k_1 = 191, k_2 = 131$ (see Table 2.4 in [7]) and the length of v is $\lceil \log_2(k_1+1) \rceil = 8$. So each parameter block can serve for $\lfloor \frac{131}{8} \rfloor = 16$ information blocks, and the efficiency of the scheme is $\frac{191}{255+255/16} = 0.7050$. But if only one fixed threshold is used, we have to use only C_2 for correcting errors. In this case, the efficiency will be $\frac{131}{255} = 0.5137$, which is much smaller than 0.7050 obtained above.

IV. Scheme Based on Balanced Codes

In the previous section, a scheme based on Berger codes is presented, where two different thresholds are used for reading data - a predetermined threshold for metadata blocks and a dynamic threshold for information blocks. More errors are introduced when reading bits based on the predetermined threshold than the adjusted dynamic one. In this section, we present another error-correcting scheme in which only dynamic thresholds are used. The idea is that we fix the number of ones in each block to be a constant. Therefore it is not necessary to store it.

We first briefly introduce balanced codes. Balanced codes, whose codewords have an equal number of ones and zeros, are widely used in storage and communication channels. Knuth proposed several beautiful algorithms for constructing balanced codes [8]. The basic idea is simple. Given an information word of k-bits, the encoder inverts the first i bits such that the modified word has an equal number of 1's and 0's. Knuth showed that such i always exists, and it is represented by a balanced word of length p. Then a codeword consists an p-bit prefix word and an k-bit modified information word. By inverting the first i bits again, the decoder can easily get the original information word. Knuth's algorithms were later improved or modified by many researchers [9] [10]. Balanced codes with error-correcting capability were also studied [11], although most of the works focus on designing balanced codes correcting at most $t \leq 4$ errors and it turns out that designing simple systematic balanced codes correcting arbitrary numbers of errors is not easy.

Considering our requirements, it is not necessary for the whole codeword to be balanced. In the scheme, we use partial balanced codes, in which only a segment of each codeword is balanced. Given a information word u of k bits, it can be efficiently represented by a balanced word x of length $m > k$. To make this balanced word have error-correcting capacity, extra parity-check bits are added by applying well-known error-correcting codes, like BCH codes. For example, we assume that each block has 255 cells and a BCH code correcting 8 errors is applied here. In this case, we have

the length of the information part $m = 191$. According to the constructions of balanced codes in [9], the length of the information word can be $k = 191 - 7 = 184$. Thus the efficiency of the scheme is $\frac{184}{255} = 0.7216$. We see it is a little more efficient than the scheme in the last section, by comparing this value with the efficiency 0.7050 obtained in the previous section. Note that in both of the cases, we used the same BCH code for the information blocks.

The reading/decoding process is also efficient. First, the threshold level v_d is obtained such that among the first m cells, there are $m/2$ cells with higher levels than v_d. Then, the whole block can be read as a binary word y of length n based on the threshold v_d, which can be further decoded as a balanced word x if the number of errors is well-bounded. Furthermore, we obtain the initial information word u following the decoding process of balanced codes. Comparing with the scheme based on Berger codes, this scheme does not have to encode or decode the metadata, hence it is a little faster for encoding and decoding.

In the scheme, the weight of each codeword is not exactly $\frac{n}{2}$ (if n is even). Instead, it can be any value in $[\frac{m}{2}, n - \frac{m}{2}]$, without constraints on the check bits of the codewords - whose weight can be any value between 0 and $n - m$. The bit error rate in storage systems is usually small. Therefore, the length of the check bits in each block is relatively small compared to the codeword length n, and the weight of each codeword in our scheme is close to $n/2$.

V. Calculation of Dynamic Thresholds

Given a block of n cells, assume their current levels are $v_1, v_2, ..., v_n$. Our problem is to determine a value v_d such that if the threshold level $v_{th} = v_d$, exactly k cells will be read as ones (i.e., with levels higher than v_d).

A trivial method is to sort all the n cell levels in decreasing order such that $v_{i_1} \geq v_{i_2} \geq ... \geq v_{i_n}$. Then $v_d = \frac{v_{i_k} + v_{i_{k+1}}}{2}$ is our desired threshold. The limitation of this method is that it needs $O(n \log n)$ computational time, which may slow down the reading speed when n is large.

A half-interval search algorithm can be used as a method with less computational complexity. Assume it is known that v_d is $\in [l_1, l_2]$ with $l_1 < l_2$. Then we can divide all the possible cell levels into three regions: $(l_2, \infty]$, $(l_1, l_2]$ and $[-\infty, l_1]$. All the cells in the first region will be read as ones and all the cells in the third region will be read as zeros. Now let k_1 be the number of cells in the first region. We continue to divide the second region into two regions $(l_3, l_2]$ and $(l_1, l_3]$, where $l_3 = \frac{l_1 + l_2}{2}$ is the middle point. We need to determine that whether $v_d \in (l_3, l_2]$ or $v_d \in (l_1, l_3]$. Let k_2 be the number of cells in the region $(l_3, l_2]$. Then there are three cases to consider:

1) If $k_2 = k - k_1$, then $v_d = l_3$ is the desired threshold value. In this case, all the cells with levels in $(l_3, l_2]$ are read as ones and those cells with levels in $(l_1, l_3]$ are read as zeros. The task of reading the block ends here.
2) If $k_2 > k - k_1$, it means v_d is located in $(l_1, l_3]$. So the cells corresponding to $(l_3, l_2]$ are read as ones. In this case, the interval-size is reduced by a factor of two by considering $(l_1, l_3]$ instead of $(l_1, l_2]$.
3) If $k_2 < k - k_1$, it means v_d is located in $(l_3, l_2]$. So the cells corresponding to $(l_1, l_3]$ are read as zeros. In this case, the interval-size is reduced by a factor of two by considering $(l_3, l_2]$ instead of $(l_1, l_2]$.

In practice, the cell levels $v_1, v_2, ..., v_n$ are read as binaries of length l with limited precisions. Repeating the above process, the interval size decreases exponentially, and finally we will obtain a value v_d satisfying the requirements of dynamic thresholds, or the interval size becomes smaller than the reading precision. In the worst case, the half-interval algorithm can be finished in $O(ln)$ time. In most cases, especially when there are not too many cells with levels very close to v_d, this algorithm can be finished in $O(n)$ time, which is faster than the method based on sorting.

VI. Conclusion

Compared to traditional fixed thresholds, dynamic thresholds have great advantages in reducing bit error probabilities especially when cell level shift is not ignorable in nonvolatile memories. In this paper, we demonstrated the performance gain of dynamic thresholds and presented two error-correcting schemes based on Berger codes and balanced codes. It can be noticed that all the methods in this paper can be generalized to multi-level memories. For example, if we apply the scheme based on Berger codes to a 4-level memory, then the metadata of each word consists of three integers, corresponding to the numbers of 1's, 2's and 3's.

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] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti. "Introduction to flash memory", *Proceedings of the IEEE*, vol. 91, pp. 489-502, 2003.
[2] A. Pirovano, A. Redaelli, et al, "Reliability study of phase-change nonvolatile memories", *IEEE Transactions on Device and Materials Reliability*, vol. 4, pp. 422-427, 2004.
[3] T. Kløve, "Error correcting codes for the asymmetric channel", *Technical Report*, Dept. of Informatics, University of Bergen, 1981. (Updated in 1995.)
[4] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck. "Codes for asymmetric limited-magnitude errors with application to multilevel flash memories", *IEEE Transactions on Information Theory*, vol. 56, no. 4, pp. 1582-1595, 2010.
[5] H. Zhou, A. Jiang, and J. Bruck. "Non-uniform Codes for Asymmetric Errors", IEEE International Symposium on Information Theory, 2011.
[6] J. M. Berger, "A note on an error detection code for asymmetric channels", *Information and Control*, vol. 4, pp. 68-73, March 1961.
[7] E. Fujiwara, "Code design for dependable systems: theory and practical applications", John Wiley & Sons, Inc., 2006.
[8] D. E. Knuth, "Efficient balanced codes", *IEEE Transactions on Information Theory*, vol. 32, no. 1, pp. 51-53, 1986.
[9] L. G. Tallini, R. M. Capocelli, and B. Bose, "Design of some new efficient balanced codes," *IEEE Transactions on Information Theory*, vol. 42, no. 3, pp. 790-802, May 1996.
[10] K. S. Immink and J. Weber, "Very efficient balanced codes", *IEEE Journal on Selected Areas in Communications*, vol. 28, pp. 188-192, 2010.
[11] S. Al-Bassam, B. Bose, "Design of efficient error-correcting balanced codes", *IEEE Transactions on Computers*, vol. 42, pp. 1261-1266, 1993.