LDPC Codes for Rank Modulation in Flash Memories

Fan Zhang
Electrical and Computer Eng. Dept.
Texas A&M University
College Station, TX 77843
fanzhang@tamu.edu

Henry D. Pfister
Electrical and Computer Eng. Dept.
Texas A&M University
College Station, TX 77843
hpfister@tamu.edu

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

***Abstract*—An LDPC code is proposed for flash memories based on rank modulation. In contrast to previous approaches, this enables the use of long ECCs with fixed-length modulation codes. For ECC design, the rank modulation scheme is treated as part of an equivalent channel. A probabilistic model of the equivalent channel is derived and a simple high-SNR approximation is given. LDPC codes over integer rings and finite fields are designed for the approximate channel and a low-complexity symbol-flipping verification-based (SFVB) message-passing decoding algorithm is proposed to take advantage of the channel structure. Density evolution (DE) is used to calculate decoding thresholds and simulations are used to compare the low-complexity decoder with sum-product decoding.**

I. Introduction

Flash memories have become the most widely used non-volatile memories (NVMs) due to their high performance. They use the charge stored in floating-gate cells to represent data. Charge (e.g., electrons) can be injected into a cell by the hot-electron injection mechanism or the Fowler-Nordheim tunneling mechanism, and be removed from the cell by the tunneling mechanism. The amount of charge in a cell is called its *level*, and we can quantize it into one of q values to store $\log_2 q$ bits. When $q = 2$, it is called a single-level cell (SLC); and when $q > 2$, it is called a multi-level cell (MLC). To increase data density, MLC flash memories with more levels (e.g., $q = 4, 8, \ldots$) are being actively developed.

Flash memories have a distinct property called *block erasure*. The flash memory cells are organized as blocks, where every block consists of about $10^5 \sim 10^6$ cells. Although the cell levels can be increased individually, to decrease any cell's level, the whole block must be erased and then reprogrammed. Block erasures significantly reduce the longevity, speed and power efficiency of flash memories. They also make cell programming (i.e., charge injection) difficult, especially for MLCs where the gap between adjacent cell levels is small, because the charge injection is a noisy process and any over-injection will lead to the expensive block erasure operation. To remove the risk of over injection and make cell programming more robust and efficient, the rank modulation scheme was proposed in [1], whereby the relative order of cell levels, instead of their absolute values, is used to represent data.

This material is based upon work supported by the National Science Foundation. The work of F. Zhang and H. Pfister was supported by Grant No. 07407470. The work of A. Jiang was supported by grant No. 0747415 and grant No. 0802107. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

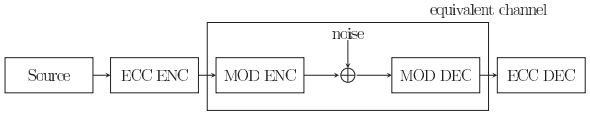

Fig. 1. Block diagram for ECC with modulation codes for flash memory.

Research has already been done on error-correcting codes (ECCs), rewriting codes and capacity analysis for rank modulation. Based on the *Kendall tau distance*, error-correcting codes for a single rank modulation block were explored in [1], [5]. In [5], a family of single-error-correcting codes was constructed, whose number of codewords is provably at least half of optimal. In [1], the asymptotic rates of optimal ECCs were derived, and the existence of good t-error-correcting codes was proved. Rewriting codes that enable data to be rewritten without block erasures and capacity analysis were also presented [4], [8].

In this paper, we study LDPC codes for rank modulation. Compared to codes designed for a single rank modulation block, where the cell levels are ordered into a permutation [1], [5], our LDPC code approach partitions the cells into many small groups and uses rank modulation separately for each group. In this work, we treat the rank modulation code as part of the channel, as shown in Fig. 1. We consider the physical channel together with the rank modulation encoder and decoder as the equivalent channel for the ECC. We analyze the equivalent channel, and study the design of good LDPC codes for the channel. A family of symbol-flipping verification-based (SFVB) decoding algorithms is proposed and analyzed.

The structure of the paper is as follows. In Section II, we study the equivalent channel model of rank modulation. Section III presents the LDPC codes design and the performance analysis for rank modulation. Section IV presents the simulation results. Section V discusses some conclusions.

II. Equivalent Channel of Rank Modulation

A. Rank Modulation

Consider a group of n cells, whose levels are $c_1, c_2, \ldots, c_n$. Here $c_i \in \mathbb{R}$ for $i = 1, \ldots, n$, and represents the amount of charge stored in the i-th cell. Let S_n be the set of

$m = n!$ permutations of length n. Specifically, we have $S_n = \{s_1, s_2, \cdots, s_m\}$ where, for $i = 1, \ldots, m$,

$$s_i \triangleq (s_i(1), s_i(2), \cdots, s_i(n))$$

is a permutation of $(1, 2, \cdots, n)$. The rank modulation scheme [4] defines a mapping $\mathcal{R}$

$$\mathcal{R} : \{(c_1, \ldots, c_n) \in \mathbb{R}^n\} \to S_n$$

as follows. If $\mathcal{R}(c_1, c_2, \ldots, c_n) = (i_1, i_2, \ldots, i_n)$, then for any $j_1 \neq j_2 \in \{1, \ldots, n\}$, $i_{j_1} > i_{j_2}$ if and only if $c_{j_1} \geq c_{j_2}$. Namely, the function $\mathcal{R}$ ranks the n cells based on the relative order of their cell levels. The rank modulation scheme uses the permutation induced by the cell levels, namely $\mathcal{R}(c_1, \ldots, c_n)$, to represent data.

Let $\mathbb{Z}_m = \{1, 2, \ldots, m\}$ be the ring of integers modulo m. Let Π be a bijection:

$$\Pi : S_n \to \mathbb{Z}_m.$$

As a modulation code, the rank modulation scheme has an encoder and a decoder, as shown in Fig. 1. Given a variable $x \in \mathbb{Z}_m$ as input to the encoder, the flash memory programs the n cells such that their cell levels $(c_1, \ldots, c_n)$ satisfy

$$\Pi(\mathcal{R}(c_1, \ldots, c_n)) = x.$$

For the decoder, it takes the cell levels $(c_1, \ldots, c_n)$ as input, and outputs the variable $\Pi(\mathcal{R}(c_1, \ldots, c_n)) \in \mathbb{Z}_m$.

B. LDPC Codes over Integer Rings for Rank Modulation

One way to design LDPC codes for rank modulation is to match the alphabet size of the LDPC codes and rank modulation. We can define the LDPC code over the integer ring $\mathbb{Z}_m$. As Fig. 1 shows. the outputs of the LDPC code encoder are mapped to permutations in S_n by Π. Then the permutation is represented by cell levels by rank modulation. The rank modulation demodulator calculates the permutations implied by the cell levels. The permutations are mapped to elements in $\mathbb{Z}_m$ for the LDPC code decoder.

An element $x' \in \mathbb{Z}_m$ has a multiplicative inverse, and is called *invertible*, if and only if x' and m are co-prime (i.e., $\gcd(x', m) = 1$). Let N be the block length of the code, and let M be the number of parity check equations. Every symbol of the code is realized by a group of n cells using rank modulation, so the code corresponds to nN cells in total. Let $\Omega \subseteq \mathbb{Z}_m$ be the subset of invertible elements which is also known as the multiplicative group of $\mathbb{Z}_m$.

An LDPC code over $\mathbb{Z}_m$ is defined by its parity-check matrix H, which is an M by N sparse matrix whose non-zero entries are chosen independently and uniformly from Ω. A valid codeword $\mathbf{X} \in \mathbb{Z}_m^N$ should satisfy all the parity check equations, i.e., $H\mathbf{X} = 0$. The number of invertible elements $|\Omega|$ is given by $\varphi(m)$, where $\varphi(\cdot)$ is Euler's totient function. It is known that large integer rings must have a reasonably large number of invertible elements because $\varphi(m)$ can be bounded by $\varphi(m) \geq m^{1-\epsilon}$ for arbitrary $\epsilon > 0$ and large enough m.

We can obtain the generator matrix G by using Gaussian elimination to place H in the row reduced form. For some parity-check matrices, the Gaussian elimination may get stuck before one finds the row-reduced echelon form of H (e.g., there is no invertible element available for exchange). However, this seems to occur with very small probability when H is sparse and m is large. The probability of finding a G from H is easily seen as equivalent to the probability of H being full rank. The results in [3] show that a uniform random matrix over $\mathbb{Z}_m$ is full rank with high probability as $m \to \infty$. Restricting our attention to sparse matrices whose non-zero entries are invertible changes the setting of the problem, and we do not have analytical results in this case. But numerical experiments show that one can almost always find a generator matrix G by trying Gaussian elimination on several randomly chosen H's.

C. LDPC Codes over Finite Fields for Rank Modulation

Another way to design LDPC codes for rank modulation is to design the codes over a finite field $GF(q)$ which is embedded into $\mathbb{Z}_m$ such that $q \leq m$ [7], [9]. In detail, let $\mathcal{L}$ be a subset of $\mathbb{Z}_{n!}$ with q elements. To associate $\mathcal{L}$ with $GF(q)$, one can design an arbitrary one-to-one correspondence between $\mathcal{L}$ and $GF(q)$ which is denoted as $\theta : \mathcal{L} \to GF(q)$.

The parity-check matrix H is an M by N matrix whose non-zero elements are randomly chosen from $GF(q)$ and the codes are defined over $GF(q)$.

The input and output alphabet sets of the encoder are both $GF(q)$ and the coded symbols are mapped to the permutations in $\mathcal{L}$ symbol by symbol by θ. Each permutation is then modulated by rank modulation and passes through the physical channel. The rank modulation demodulator calculates the permutation implied by the cell levels read back from the channel and the permutation is mapped back to $GF(q)$ as the input of the LDPC code decoder. Note that sometimes the output of the rank modulation demodulator is not in $\mathcal{L}$ due to the channel error. In this case, the outputs of the rank modulation demodulator need to be pre-processed before LDPC code decoding. We will discuss this detail more in next section.

Comparing to the codes over integer rings described in previous subsection, here we only use q symbols out of all m possible symbols implied by rank modulation during the encoding and this causes a rate loss by a factor of $\zeta_n \triangleq \frac{\log q}{\log n!}$. For example, $\zeta_5 = 0.869$, $\zeta_6 = 0.948$, $\zeta_7 = 0.976$ and $\zeta_8 = 0.980$. But, this field-embedding modification also leads to codes with lower false verification probability and better performance in the error floor regime [7], [9].

D. Equivalent Channel

In this subsection, we consider the equivalent channel for rank modulation. By treating the modulation encoder and decoder as part of the channel, the input and output alphabet sets of the equivalent channel are both $\mathbb{Z}_m$. Let the input variable be x, and the output variable be y, where $x, y \in \mathbb{Z}_m$. We first derive the transition probability of the equivalent channel $W(y|x)$. Due to the difficulty of having a closed-form

of $W(y|x)$, we derive an approximation of this channel when the noise is i.i.d. additive Gaussian with small variance.

Let us first consider the general case. Consider a group of n cells, and let the physical noise in the cell levels have the joint probability density function (pdf) $f(w_1, w_2, \cdots, w_n)$, where w_i is the noise in the i-th cell's level. The rank-modulation encoder maps the input integer x to $s_i = \Pi^{-1}(x)$, and programs the cell levels as $c = \{c_1, c_2, \cdots, c_n\}$ such that $\mathcal{R}(c) = s_i$. Then the cell levels are distorted by the additive noise $w = (w_1, w_2, \cdots, w_n)$, where w_i is the noise in the i-th cell level. The rank-modulation decoder reads the noisy cell levels $c' = \{c_1 + w_1, c_2 + w_2, \cdots, c_n + w_n\}$, computes the permutation $s_j = \mathcal{R}(c')$, and outputs the integer $y = \Pi(s_j)$. (Note that the mapping $\Pi(\cdot)$ can be chosen arbitrarily.) Without loss of generality, we assume that $\Pi^{-1}(1) = (1, 2, \cdots, n)$, the identity permutation. Since Π is a bijection, we also abuse notation and write $W(s_j|s_i) = W(y|x)$ for a channel whose inputs and outputs are permutations.

Let π be a permutation on $\{1, 2, \ldots, n\}$. Since permutations naturally operate on each other via composition, we have $\pi \circ s_i = (\pi(s_i(1)), \ldots, \pi(s_i(n)))$. For any fixed bijection Π, the equivalent channel is *symmetric* if and only if for all s_i and s_j, $W\left(\Pi(s_j)|\Pi(s_i)\right) = W\left(\Pi(\pi \circ s_j|\Pi(\pi \circ s_i)\right)$ holds for all π. When the noise is i.i.d., it is easily seen that the equivalent channel is symmetric. Therefore, without loss of generality, we assume the input variable is 1, and analyze $W(y|1)$. For any input integer x, the channel transition probability is $W(y|x) = W(y'|1)$ where $y' = \Pi(\pi \circ \Pi^{-1}(y))$, and π is the unique permutation satisfying $\pi \circ \Pi^{-1}(x) = \Pi^{-1}(1)$.

Let $s_1 = \Pi^{-1}(1) = (1, 2, \cdots, n)$, $c = (1, 2, \ldots, n)$, and $c' = c + w$. Then, the hard decision receiver for rank modulation computes $y = \mathcal{R}(c') = \mathcal{R}((c_1 + w_1, \ldots, c_n + w_n))$ and the channel transition probability $W(s_j|s_1)$ is given by

$$W(s_j|s_1) = \int_{\mathcal{A}_j} f(w_1, w_2, \cdots, w_n) \mathrm{d}w_1 \mathrm{d}w_2 \cdots \mathrm{d}w_n$$

where $\mathcal{A}_j = \mathcal{R}^{-1}(s_j)$ is the decision region for s_j. The integration domain $\mathcal{A}_j$ can be simplified to the intersection of $n - 1$ half spaces given by

$$\mathcal{A}_j \triangleq \left\{ w \;\middle|\; \begin{array}{c} w_{s_j(1)} + c_{s_j(1)} < w_{s_j(2)} + c_{s_j(2)} \\ w_{s_j(2)} + c_{s_j(2)} < w_{s_j(3)} + c_{s_j(3)} \\ \vdots \\ w_{s_j(n-1)} + c_{s_j(n-1)} < w_{s_j(n)} + c_{s_j(n)} \end{array} \right\}.$$

This decision region remains valid for the general case where the cell levels are not equally separated and the noise has an arbitrary joint distribution on the n cells. Since the integral cannot be further simplified without making more assumptions, we compute it numerically using Monte Carlo integration.

In the following, we assume that the noise is i.i.d. Gaussian with zero mean and small variance σ^2. We also assume that all the adjacent cell levels are separated equally by a constant Δ. (Note that for rank modulation, the memory only needs to use comparison circuits to program and read cells, and the real cell levels are unknown to the decoder. Results obtained by assuming evenly-spaced cell levels can be extended later for more robust code design.)

To study the approximate channel transition probability $\tilde{W}(y|x)$, we first define two concepts.

Definition 1: An *adjacent transposition* in a rank-modulation symbol is a transposition of adjacent cell levels (among the n cells). For example, an adjacent transposition in a permutation induced by the n cell levels, causes the two integers i and $i+1$ (for some $i \in \{1, 2, \ldots, n-1\}$) to switch their positions.

Definition 2: For $i = 0, 1, \ldots, \frac{n(n-1)}{2}$, for any permutation $s \in S_n$, we define $\mathcal{N}_i(s)$, called the i-th neighborhood of s, as follows: the minimum number of adjacent transpositions to change any permutation in $\mathcal{N}_i(s)$ to s (and vice versa) is i. Namely, $\mathcal{N}_i(s)$ is the set of permutations at Kendall tau distance i from s. We note that $|\mathcal{N}_1(s)| = n - 1$.

Given the input symbol $s \in S_n$ to the channel, the output symbol will be s with probability $1 - p$. Let p_1 be the approximate probability that the output symbol is $s' \in \mathcal{N}_1(s)$. For small noise, it can be shown that $p \approx |\mathcal{N}_1(s)|p_1 = (n-1)p_1$, that is, the dominant errors are the errors causing one adjacent transposition. The following lemma makes this notion precise, Due to the limited space, its proof is omitted.

Lemma 1: For additive i.i.d. Gaussian noise with small variance σ^2, the channel model can be approximated as follows. (We call it the *approximate channel* model.) Let $x, y \in S_n$ be the input and output permutations to the channel, respectively. Then the channel transition probability for the approximate channel is

$$\tilde{W}(y|x) = \begin{cases} 1 - (n-1)p_1 + o(p_1) & \text{if } x = y \\ p_1 + o(p_1) & \text{if } y \in \mathcal{N}_1(x) \\ O(p_1^2) & \text{otherwise} \end{cases} \quad (1)$$

where $p_1 = \frac{\sigma}{\sqrt{\pi}\Delta} \exp\left(-\frac{\Delta^2}{4\sigma^2}\right)$.

III. LDPC Codes Design for Rank Modulation

A. Iterative verification-based decoder

We are interested in message-passing decoders for this problem because they have complexity that is linear in N. In particular, we design a *symbol-flipping* algorithm based on verification. The idea is to jointly use the a priori information about the channel and the assumption that two error messages match with low enough probability as n is large enough. We note that this general idea can be applied to the decoders of both codes over integer rings and finite fields and the density evolution analysis is exactly the same for both codes. This results in decoding algorithms that are similar to both symbol-flipping algorithms, which can be seen as a generalization of the Gallager-A and Gallager-B hard decision decoding algorithms [2], and verification-based decoding algorithm [6]. We also refer these algorithms as the symbol-flipping verification-based (SFVB) algorithms. The event that two incorrect symbols match is called false verification (FV). The analysis of VB decoding algorithms typically considers DE with no FV and the probability of FV separately.

Note that the verification-based decoding algorithms can be further classified into node-based (NB) and message-based (MB) algorithms [9]. Therefore, the SFVB decoding algorithms also have two categories, namely, node-based SFVB (NBSFVB) decoding algorithms and message-based SFVB (MBSFVB) decoding algorithms. The messages from the variable nodes to the check nodes (resp. from the check nodes to the variable nodes) are multiplied by the (resp. inverse of the) random entries in the parity-check matrix. For simplicity, we will not describe this explicitly in the following.

1) NBSFVB decoding algorithm: In each iteration, we first calculate the check sums of the check nodes. Then we perform the symbol-flipping operation on the variable nodes simultaneously by the following rules.

- If all the check nodes connected to a variable node are satisfied (i.e., the check sum equals 0), then the variable node remains at its current value x.
- Otherwise, the variable node tries to change its value to one of the $\mathcal{N}_1(y)$ neighbors of its received value y.
  - If it finds a value $x' \in \mathcal{N}_1(y)$ which satisfies all the checks, then the variable node flips its value to x': $x \to x'$.
  - If it cannot find a value which satisfies all the checks, it keeps its value unchanged.

We note that this algorithm cannot be analyzed by density evolution since the two-way messages on the same edge are not independent. This algorithm is a node-based message-passing decoding [9] and one can analyze it using the differential equation approach described in [7] [9]. Also note that this algorithm can be further improved by relaxing the flipping condition. The idea is similar to Gallager-B hard decision decoding algorithm [2] for the BSC and we skip the details.

2) MBSFVB decoding algorithm : By using the a priori channel information and the fact that the size of the channel output alphabet set is large, we propose the MBSFVB decoding algorithm. The decoding rules are described as follows.

- Check Node Operation: Let the variable-to-check messages be $m_0, m_1, \cdots, m_{r-1}$. For a check node of degree r, the check-to-variable message on the j-th edge is $m'_j = m_j - \sum_{k=0}^{r-1} m_k$.
- Variable Node Operation: For a variable node of degree l, let the channel output value be y and check-to-variable messages be $m'_0, m'_1, \cdots, m'_{l-1}$. The variable-to-check message on the j-th edge is

$$m_j = \begin{cases} m, & \text{if C1} \\ y' & \text{if not(C1) and C2} \\ y & \text{otherwise} \end{cases}$$

  where condition C1 represents the event that at least 2 messages from the message set $\{y\} \cup \{m'_0, m'_1, \cdots, m'_{l-1}\} \backslash \{m'_j\}$ match and equal m. Condition C2 represents the event that there exists a value $y' \in \mathcal{N}_1(y)$ such that y' matches at least 1 message in $\{m'_0, m'_1, \cdots, m'_{l-1}\} \backslash \{m'_j\}$.

Based on [9], we derive the DE analysis of MBSFVB under the assumption that the output message is correct whenever condition C1 or C2 holds (i.e., there is no false verification (FV)). Let p_i be the probability that a variable-to-check message is incorrect at iteration i. Let q_i be the probability that a check-to-variable message is incorrect at iteration i. Note that $p_0 = p$ is the channel error probability. The DE equations for the MBSFVB algorithm are $q_i = 1 - (1 - p_i)^{r-1}$ and $p_{i+1} = p q_i^{l-1}$.

Taking the irregularity into account, the DE equations can be written as $q_i = 1 - \rho(1 - p_i)$ and $p_{i+1} = p\lambda(q_i)$, where $\lambda(x)$ and $\rho(x)$ are the variable node and check node degree distributions in the edge perspective.

Remark 1: The DE recursion of the MBSFVB algorithms is identical to the DE recursion of the BEC. The decoding threshold of the (3,6) ensemble with the MBSFVB algorithm is $p^*_{MBSFVB} = 0.428$. The decoding threshold of the (3,50) ensemble with the MBSFVB algorithm is $p^*_{MBSFVB} = 0.047$.

As mentioned in previous section, when the codes are defined over $GF(q)$ which is embedded into $\mathbb{Z}_m$, sometimes the output permutation y of the rank modulation demodulator is not in $\mathcal{L}$, hence y cannot be mapped back to $GF(q)$. In this case, we know that the correct permutation falls in $\mathcal{N}_1(y)$ with high probability. We assign an arbitrary element in $Z(y) \triangleq \{z | z \in GF(q), \theta^{-1}(z) \in \mathcal{N}_1(y), \theta^{-1}(z) \in \mathcal{L}\}$ to this coded symbol and we assign the rest elements in $Z(y)$ as its decoding neighborhood. Then we can still apply the SFVB decoding algorithms described above. We note that this results in slightly better error correcting performance than that predicted by the DE analysis because there is less ambiguity in those symbols on the boundary of $\mathcal{L}$ given the channel observation.

Since the error correcting capability is better if the transmitted symbol is on the boundary of $\mathcal{L}$, the analysis of the average performance cannot be simplified to the analysis of the all-1 codeword (where each permutation is the identity permutation). We only use the DE result as an upper bound of the decoding threshold.

In our DE analysis, we assume that there is no FV. During our simulations of codes over $\mathbb{Z}_m$, we discovered that the probability of FV is not negligible for moderate n (e.g., $n \leq 8$). The problem is not that the alphabet size is too small (e.g., $8! = 40320$ is large enough based on previous work). Instead, it appears that the issue of FV is more complicated for codes over integer rings. The reason is that multiplication with random edge weights maps some incorrect symbols (e.g., $(n-1)! \in \mathbb{Z}_m$) to sets whose size are significantly smaller than m or even $\phi(m)$. Therefore, the probability that two incorrect symbols match is too large to be ignored. Unfortunately, we do not have a good analysis of the FV probability. Still, the simulation results show that, for $n \geq 8$, the probability of FV is low enough that the algorithm may be useful.

IV. Simulation Results

We evaluate the capacity of the real channel and the approximate channel. The result is shown in Fig. 2. From the result we can see that the approximate channel and the real channel have approximately the same capacity when $p < 0.05$.

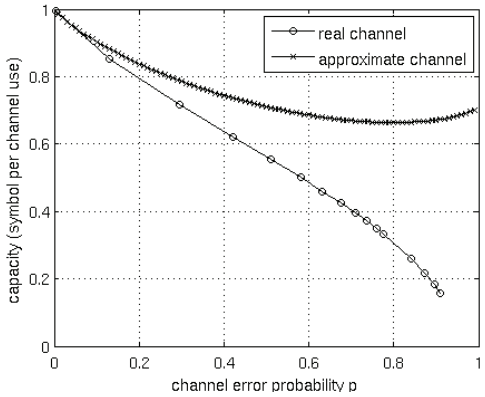

Fig. 2. Capacity of the real channel model and the approximate channel.

We simulate the MBSFVB decoder, or the verification-based (VB) decoder, with the $(3, 50)$ regular ensemble with rate 0.94 and threshold $p^*_{VB} = 0.047$. The reason we choose the $(3, 50)$ ensemble is that the approximate channel and the real channel model have approximately the same capacity around the decoding threshold. The channel used is the real channel model. Note that the threshold $p^*_{VB} = 0.047$ implies the threshold $\sigma^*_{VB} = 0.315$ in the real channel model. The codes are generated without 4 cycles. Each coded symbol is stored by 5 to 8 cells with rank modulation. We simulate both codes over integer rings and codes over finite fields with different alphabets. We also simulate the VB decoder without FV by artificially avoiding FV's. To see how the numerical simulation matches the DE analysis, we choose the block length to be 10^5 and the maximum number of decoding iterations to be 100. Each point is the average of up to 10^8 trials. And the results are shown in Fig. 3. It can be seen that the simulation matches the DE analysis very well.

To be complete, we also compare the VB decoder with the full belief propagation (BP) decoder, where each message is a probability mass function (pmf) over $\mathbb{Z}_m$. For the check node, the check-to-variable message is calculated by the convolution of all other variable-to-check pmf's. For the variable node, the variable-to-check message is calculated by normalizing the product of the channel pmf and all other check-to-variable pmf's. The complexity of this method in the probability domain is $O(Nm^2)$ per iteration without optimization. The log domain FFT-based decoder has complexity $O(Nm\log(m))$ per iteration [10], [11].

We compare the VB decoder with the full BP decoder in Fig. 3. We simulate codes from the (3,50) ensemble by the VB decoder and the full BP decoder. We simulate codes over integer rings $\mathbb{Z}_m$ and finite fields which are embedded in $\mathbb{Z}_m$. We choose the block length to be 1000 and $n = 5$ to 8 and without FV for the VB decoder and $n = 5$ for the full BP decoder due to due to its large complexity. The real channel model is used and the x-axis is the standard deviation σ of the i.i.d. Gaussian noise. From the results, we can see the VB decoder without FV's has similar performance with the full decoder. The performance loss of the codes over $\mathbb{Z}_m$ with $n = 5$ and $n = 8$ is due to the probability of FV's. This loss

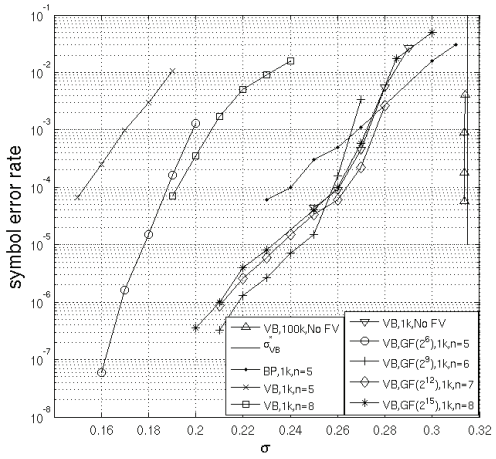

Fig. 3. Simulation results comparing the VB decoder with the full BP decoder using the real channel model.

can be compensated by using the codes over $GF(q)$ which is embedded into the rings. This improves the performance significantly with a slight rate loss.

V. Conclusion

In this paper, we consider LDPC codes as error correcting codes (ECC's) for rank modulation in flash memories. We treat rank modulation as part of the equivalent channel for the ECC, and analyze the probabilistic model of this channel and a high-SNR approximation. We design LDPC codes over integer rings and finite fields together with a family of symbol-flipping verification-based (SFVB) decoding algorithms to achieve good performance with low-complexity. Simulation results are used to verify the analysis and compare with more complicated decoders.

References

[1] A. Barg and A. Mazumdar, "Codes in permutations and error correction for rank modulation," Available as Arxiv preprint cs.IT/0908.4094, 2009.
[2] R. G. Gallager, *Low-density parity-check codes*, the M.I.T. press, Cambridge, MA, USA, 1963.
[3] R. P. Brent and B. D. McKay, "Determinants and ranks of random matrices over Z_m," in *Discrete Mathematics*, vol. 66, pp. 35-49, 1987.
[4] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank modulation for flash memories," in *IEEE Trans. Information Theory*, vol. 55, no. 6, pp. 2659-2673, 2009.
[5] A. Jiang, M. Schwartz and J. Bruck, "Error-correcting codes for rank modulation," in *Proc. IEEE ISIT*, 2008, pp. 1736-1740.
[6] M. Luby and M. Mitzenmacher, "Verification-based decoding for packet-based low-density parity-check codes," in *IEEE Trans. Information Theory*, vol. 51, no. 1, pp. 120-127, 2005.
[7] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi and D. A. Spielman, "Efficient erasure correcting codes," in *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 569-584, Feb. 2001.
[8] Z. Wang, A. Jiang and J. Bruck, "On the capacity of bounded rank modulation for flash memories," in *Proc. IEEE ISIT*, 2009, pp. 1234-1238.
[9] F. Zhang and H. D. Pfister, "List-message passing achieves capacity on the q-ary symmetric channel for large q," submitted to *IEEE Trans. Information Theory*. Available as Arxiv preprint cs.IT/0806.3243, 2008.
[10] H. Song and J. R. Cruz, "Reduced-complexity decoding of Q-ary LDPC codes for magnetic recording," in *IEEE Trans. Magnetics*, vol. 39, no. 3, pp. 1081-1087, Mar. 2003.
[11] M. Davey and D. J. C. MacKay, "Low density parity check codes over GF(q)," in *IEEE Commun. Lett.*, vol. 2, no. 6, pp. 165-167, Jun. 1998.