Introduction to non-volatile memories

What is a non-volatile memory?

• Mismatch in meanings

Wiki:

• *Non-volatile memory is computer memory that can get back stored information even when not powered.*

What people really mean:

• Solid-state memories (electrically addressed)
• Exclude hard-disks, optical disks, etc. such that the storage is mechanically addressed
What is a non-volatile memory?

• Mismatch in meanings

Wiki:

• *Non-volatile memory is computer memory that can get back stored information even when not powered.*

What people really mean:

• Solid-state memories (*electrically addressed*)
• Exclude hard-disks, optical disks, etc. such that the storage is *mechanically addressed*
Introduction to non-volatile memories

What is a non-volatile memory?

• Examples of non-volatile memory
  • Flash memory (most widely-used)
  • Phase-change memory
  • Several RAMs
Introduction to non-volatile memories

- Flash memories
Introduction to non-volatile memories

- Flash memories
Outline

• Modulation codes
  • ICI-free codes for flash memories
  • Rewriting codes (write-once memories codes)
  • Constrained codes for phase-change memories

  • Error correction codes
    • Erasure codes for distributed storage
    • Belief-propagation decoding of polar codes

• Read/write strategy for flash memories
  • Read thresholds estimation
  • Parallel programming of flash memories
    • Discrete-level representation
    • Rank modulation
Outline

• Modulation codes
  • ICI-free codes for flash memories
  • Rewriting codes (write-once memories codes)
  • Constrained codes for phase-change memories

• Error correction codes
  • Erasure codes for distributed storage
  • Belief-propagation decoding of polar codes

• Read/write strategy for flash memories
  • Read thresholds estimation
  • Parallel programming of flash memories
    • Discrete-level representation
    • Rank modulation
Outline

• Modulation codes
  • ICI-free codes for flash memories
  • Rewriting codes (write-once memories codes)
  • Constrained codes for phase-change memories
• Error correction codes
  • Erasure codes for distributed storage
  • Belief-propagation decoding of polar codes
• Read/write strategy for flash memories
  • Read thresholds estimation
  • Parallel programming of flash memories
    • Discrete-level representation
    • Rank modulation
Outline

- **Modulation codes**
  - ICI-free codes for flash memories
  - Rewriting codes (write-once memories codes)
  - Constrained codes for phase-change memories
- **Error correction codes**
  - Erasure codes for distributed storage
  - Belief-propagation decoding of polar codes
- **Read/write strategy for flash memories**
  - Read thresholds estimation
  - Parallel programming of flash memories
    - Discrete-level representation
    - Rank modulation
Modulation codes

Physical properties of non-volatile memories in read/write

- Physical constraints
  - Mathematical model
  - Constrained systems
  - Capacity
  - Code design
Modulation codes

Physical properties of non-volatile memories in read/write

• Physical constraints
  → Mathematical model
     → Constrained systems
     → Capacity
     → Code design
Modulation codes

Physical properties of non-volatile memories in read/write

- Physical constraints
  - Mathematical model
  - Constrained systems
  - Capacity
  - Code design
Modulation codes

Physical properties of non-volatile memories in read/write

• Physical constraints
  ➔ Mathematical model
  ➔ Constrained systems
  ➔ Capacity
  ➔ Code design
Modulation codes

Physical properties of non-volatile memories in read/write

• Physical constraints
  → Mathematical model
  → Constrained systems
  → Capacity
  → Code design
Modulation codes

Physical properties of non-volatile memories in read/write

• Writing flash memory cells
  • Inter-cell interference
  • Block erasures required when decreasing cell levels
  * Reading flash memory cells
    * Voltage threshold drift due to charge leakage
  * Writing phase-change memories
    * Heat accumulation
    * Crosstalk — Inter-cell interference
Modulation codes

Physical properties of non-volatile memories in read/write

• Writing flash memory cells
  • Inter-cell interference
  • Block erasures required when decreasing cell levels
• Reading flash memory cells
  • Voltage threshold drift due to charge leakage
• Writing phase-change memories
  • Heat accumulation
  • Crosstalk --- Inter-cell interference
Modulation codes

Physical properties of non-volatile memories in read/write

- Writing flash memory cells
  - Inter-cell interference
  - Block erasures required when decreasing cell levels
- Reading flash memory cells
  - Voltage threshold drift due to charge leakage

- Writing phase-change memories
  - Heat accumulation
  - Crosstalk --- Inter-cell interference
Outline

• Modulation codes
  • ICI-free codes for flash memories
  • Rewriting codes (write-once memories codes)
  • Constrained codes for phase-change memories
  • Error correction codes
    • Erasure codes for distributed storage
    • Belief-propagation decoding of polar codes
  • Read/write strategy for flash memories
    • Read thresholds estimation
    • Parallel programming of flash memories
      • Discrete-level representation
      • Rank modulation
ICI-free codes for flash memories

Physical constraint $\rightarrow$ Mathematical model

- Mitigate inter-cell interference $\rightarrow$ 101 is forbidden
- In general, $(q-1, 0, q-1)$ is forbidden for $q$-ary cells

‘101’ not welcomed!
ICI-free codes for flash memories

Physical constraint $\rightarrow$ Mathematical model $\rightarrow$ Constrained system

- Mitigate inter-cell interference $\rightarrow$ 101 is forbidden
- In general, $(q-1, 0, q-1)$ is forbidden for $q$-ary cells
ICI-free codes for flash memories

Physical constraint → Mathematical model → Constrained system → Capacity & Code design

**Definitions:**
Let $N_S(n)$ be the number of sequences of length $n$ satisfying the constraint $S$, then the *capacity* of the constraint is

$$C(S) = \lim_{n \to \infty} \frac{\log(N_S(n))}{n}$$

**Examples:**
- Binary: $C(\text{no-101}) \approx 0.8114$ bits/cell
- Ternary: $C(\text{no-202}) \approx 1.5327$ bits/cell

Center for Magnetic Recording Research (CMRR)
University of California, San Diego
ICI-free codes for flash memories – read cycles

• Charge Leakage $\rightarrow$ voltage drift in one direction
• Fixed threshold vs. dynamic threshold

![Voltage distribution at time 0 and time T with different thresholds for 0 and 1 states]
ICI-free codes for flash memories – read cycles

- Charge Leakage $\rightarrow$ voltage drift in one direction
- Fixed threshold vs. dynamic threshold

![Diagram showing voltage distribution at time 0 and time T](image)

At time $T$
ICI-free codes for flash memories – read cycles

• During programming, half of cells store 0 and the other half store 1 --- Balanced codes
  * During reading, those \( n/2 \) cells with lower voltages are read as 0’s, and the other \( n/2 \) cells with higher voltages are read as 1’s
  * Relative ranking might be preserved
ICI-free codes for flash memories – read cycles

• During programming, half of cells store 0 and the other half store 1 — Balanced codes
• During reading, those $n/2$ cells with lower voltages are read as 0’s, and the other $n/2$ cells with higher voltages are read as 1’s
• Relative ranking might be preserved
ICI-free codes for flash memories – read cycles

- During programming, half of cells store 0 and the other half store 1 --- Balanced codes
- During reading, those $n/2$ cells with lower voltages are read as 0’s, and the other $n/2$ cells with higher voltages are read as 1’s
- Relative ranking might be preserved
ICI-free codes for flash memories – read cycles

• During programming, half of cells store 0 and the other half store 1 --- Balanced codes

• During reading, those \( n/2 \) cells with lower voltages are read as 0’s, and the other \( n/2 \) cells with higher voltages are read as 1’s

• Relative ranking might be preserved
ICI-free codes for flash memories – read cycles

Combination of constraints: (ICI-free & balanced) codes

• Let $N_{101\text{+bal}}(n)$ be the number of binary \textit{balanced} sequences of length $n$ that \textit{forbid} 101.

\textbf{Problem:}

• What is $N_{101\text{+bal}}(n)$?

• What is $C_{101\text{+bal}} = \lim_{n \to \infty} \frac{\log(N_{101\text{+bal}}(n))}{n}$?
ICI-free codes for flash memories – read cycles

**Bijection:**

\{101-free balanced sequences\} \leftrightarrow \{UDU-free symmetric paths\}

(110001) \leftrightarrow (110001)

(111000) \leftrightarrow (111000)

(110010) \leftrightarrow (110010)

(100110) \leftrightarrow (100110)

(100011) \leftrightarrow (100011)
ICI-free codes for flash memories – read cycles

Theorem:

\[ N_{101+\text{bal}}(n) = 2 \sum_{j=0}^{(n/2)-1} \binom{j}{j/2} \left(\frac{n}{2} - 1\right), \quad n \text{ even} \]

The generating function of \( N_{101+\text{bal}}(n) \) is

\[ \varphi(x) = \sqrt{\frac{1 + x^2}{1 - 3x^2}} - 1 \]

and,

\[ C_{101+\text{bal}} = \lim_{n \to \infty} \frac{\log_2\left(N_{101}(n)\right)}{n} = \frac{1}{2} \log_2 3 \]
ICI-free codes for flash memories – read cycles

**Definition 6.** Let $S$ be a constrained system over $\Sigma$ presented by an irreducible deterministic graph $G = (V, E, L)$. Denote by $\Delta(G)$ the set of all stationary Markov chains on $G$. The entropy of $\mathcal{P} \in \Delta(G)$ is denoted by $H(\mathcal{P})$.

Given a stationary Markov chain $\mathcal{P} \in \Delta(G)$, along with a vector of real-valued functions $f = (f_1, f_2, \ldots, f_t) : E_G \rightarrow \mathbb{R}^t$, we denote by $E_{\mathcal{P}}(f)$ the expected value of $f$ with respect to $\mathcal{P}$, i.e.,

$$E_{\mathcal{P}}(f) = \sum_{e \in E_G} \mathcal{P}(e)f(e).$$

For a given symbol subset $W$ of size $t$ and a vector $p \in [0, 1]^t$, we now define the quantity

$$S_W(p) = \sup_{\mathcal{P} \in \Delta(G)} H(\mathcal{P}) \quad \text{such that} \quad E_{\mathcal{P}}(\mathcal{I}_W) = p$$

Capacity of local + global constraints (e.g., no-101 + balanced)
ICI-free codes for flash memories – read cycles

**Theorem:**

\[
\sup_{\mathcal{P} \in \Delta(G)} H(\mathcal{P}) = \inf_{x \in \mathbb{R}^t} \left\{ x \cdot r + \log \lambda(A_G; f(x)) \right\},
\]

\[
E_P(f) = r
\]

where \(\lambda(A)\) is the Perron-Frobenius eigenvalue of the nonnegative square matrix \(A\).
ICI-free codes for flash memories – read cycles

Results:

• Binary: (2% reduction)

\[ C_{101} \approx 0.8114 \]
\[ C_{101+\text{bal}} = \frac{1}{2} \log_2 3 \approx 0.7925 \]

• Ternary: (0.45% reduction)

\[ C_{202} \approx 1.5327 \]
\[ C_{202+\text{bal}} \approx 1.5258 \]
Outline

• Modulation codes
  • ICI-free codes for flash memories
  • Rewriting codes (write-once memories codes)
  • Constrained codes for phase-change memories

• Error correction codes
  • Erasure codes for distributed storage
  • Belief-propagation decoding of polar codes

• Read/write strategy for flash memories
  • Read thresholds estimation
  • Parallel programming of flash memories
    • Discrete-level representation
    • Rank modulation
Rewriting codes

- Physical constraint $\rightarrow$ Mathematical model
- Block erasure: time consuming + lifetime reducing
- Solution:
  - Write-one memories (WOM) codes
Rewriting codes

- Introduced by Rivest and Shamir, “How to reuse a write-once memory”, 1982
- The memory elements represent bits (2 levels) and can be irreversibly programmed from the ‘0’ state to the ‘1’ state

**Problem:**
- What is the maximum total number of bits that can be written using $n$ cells and $t$ writes?
Rewriting codes

• Introduced by Rivest and Shamir, “How to reuse a write-once memory”, 1982
• The memory elements represent bits (2 levels) and can be irreversibly programmed from the ‘0’ state to the ‘1’ state
• Problem:
  • What is the maximum total number of bits that can be written using \( n \) cells and \( t \) writes?

\[
R_i(n) = \frac{\log_2 M_i}{n}, \quad i = 1, \ldots, t
\]

\[
C_{\text{sum}}(t) = \lim_{n \to \infty} \sum_{i=1}^{t} R_i(n)
\]
Rewriting codes

**Example:** \( n = 3 \) cells storing 4 messages (2 bits) twice \( (t = 2) \),
sum-rate = 2 bits \( \times \) 2 writes/3 cells = 4/3 bits/cells

<table>
<thead>
<tr>
<th>data</th>
<th>1(^{st}) write</th>
<th>2(^{nd}) write</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>111</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>011</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>001</td>
<td>110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Messages on both writes</th>
<th>Cell levels after 1(^{st}) write</th>
<th>Cell levels after 2(^{nd}) write</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 3)</td>
<td>(100)</td>
<td></td>
</tr>
<tr>
<td>(2, 2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: $n = 3$ cells storing 4 messages (2 bits) twice ($t = 2$), sum-rate = $2 \text{ bits} \times 2 \text{ writes/3 cells}$ = $4/3 \text{ bits/cells}$

<table>
<thead>
<tr>
<th>data</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; write</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; write</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>111</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>011</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>001</td>
<td>110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Messages on both writes</th>
<th>Cell levels after 1&lt;sup&gt;st&lt;/sup&gt; write</th>
<th>Cell levels after 2&lt;sup&gt;nd&lt;/sup&gt; write</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 3)</td>
<td>(100)</td>
<td>(110)</td>
</tr>
<tr>
<td>(2, 2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Rewriting codes**

**Example:** \( n = 3 \) cells storing 4 messages (2 bits) twice (\( t = 2 \)), sum-rate = 2 bits \( \times \) 2 writes/3 cells = 4/3 bits/cells

<table>
<thead>
<tr>
<th>data</th>
<th>1(^{st}) write</th>
<th>2(^{nd}) write</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>111</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>011</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>001</td>
<td>110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Messages on both writes</th>
<th>Cell levels after 1(^{st}) write</th>
<th>Cell levels after 2(^{nd}) write</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 3)</td>
<td>(100)</td>
<td>(110)</td>
</tr>
<tr>
<td>(2, 2)</td>
<td>(010)</td>
<td></td>
</tr>
</tbody>
</table>
Rewriting codes

Example: \( n = 3 \) cells storing 4 messages (2 bits) twice \((t = 2)\),
sum-rate = 2 bits \( \times 2 \) writes/3 cells
= 4/3 bits/cells

<table>
<thead>
<tr>
<th>data</th>
<th>1(^{st}) write</th>
<th>2(^{nd}) write</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>111</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>011</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>101</td>
</tr>
<tr>
<td>3</td>
<td>001</td>
<td>110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Messages on both writes</th>
<th>Cell levels after 1(^{st}) write</th>
<th>Cell levels after 2(^{nd}) write</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 3)</td>
<td>(100)</td>
<td>(110)</td>
</tr>
<tr>
<td>(2, 2)</td>
<td>(010)</td>
<td>(010)</td>
</tr>
</tbody>
</table>
Rewriting codes

Example: \( n = 3 \) cells storing 4 messages (2 bits) twice \((t = 2)\), sum-rate = 2 bits \(\times\) 2 writes/3 cells
\[= \frac{4}{3} \text{ bits/cells}\]

Theorem: [Wolf, Wyner, Ziv, Korner ’84] [Heegard ’85], [Fu, Han Vinck ’99]
The sum-capacity of binary WOM is
\[C_{\text{sum},2}(t) = \log_2(t + 1).\]
The sum-capacity of \(q\)-ary WOM (cell-levels can only \(\uparrow\)) is
\[C_{\text{sum},q}(t) = \log_2\left(\binom{t + q - 1}{q - 1}\right)\]
Rewriting codes

- 2 cells, 2 writes, 8 levels for each cell
- Levels can only increase

**Example:**
- 1\textsuperscript{st} write, write to levels (2, 5)
- 2\textsuperscript{nd} write, write to levels (4, 6)
Rewriting codes – Lattice-based WOM

- 2 cells, 2 writes, 8 levels for each cell
- Levels can only increase

**Rate pair**
- \( M_1 = \{1, \ldots, 24\} \)
- \( M_2 = \{1, \ldots, 23\} \)

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>Cell-level changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>15</td>
<td>(0,0) ( \rightarrow ) (3,3) ( \rightarrow ) (6,7)</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>(0,0) ( \rightarrow ) (5,0) ( \rightarrow ) (7,7)</td>
</tr>
</tbody>
</table>

\[ \text{Sum-rate} = \frac{1}{2} (\log_2(23) + \log_2(24)) \]
\[ \approx 4.554 \text{ bits/cell} \]

\[ \text{Sum-rate without WOM} = 3 \text{ bits/cell} \]
Rewriting codes – Lattice-based WOM

- 2 cells, 2 writes, 8 levels for each cell
- Levels can only increase

Rate pair
- \( M_1 = \{1, \ldots, 24\} \)
- \( M_2 = \{1, \ldots, 23\} \)

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>Cell-level changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>15</td>
<td>(0,0) (\rightarrow) (3,3) (\rightarrow) (6,7)</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>(0,0) (\rightarrow) (5,0) (\rightarrow) (7,7)</td>
</tr>
</tbody>
</table>

Sum-rate = \(\frac{3}{2} (\log_2(23) + \log_2(24)) \approx 4.554 \text{ bits/cell} \)

Sum-rate without WOM = 3 bits/cell

Center for Magnetic Recording Research (CMRR)
Rewriting codes – Lattice-based WOM

- 2 cells, 2 writes, 8 levels for each cell
- Levels can only increase

**Rate pair**
- \( M_1 = \{1, ..., 24\} \)
- \( M_2 = \{1, ..., 23\} \)

<table>
<thead>
<tr>
<th>1\textsuperscript{st}</th>
<th>2\textsuperscript{nd}</th>
<th>Cell-level changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>15</td>
<td>(0,0) (\rightarrow) (3,3) (\rightarrow) (6,7)</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>(0,0) (\rightarrow) (5,0) (\rightarrow) (7,7)</td>
</tr>
</tbody>
</table>

\[
\text{Sum-rate} = \frac{1}{2} (\log_2(23) + \log_2(24)) \\
\approx 4.554 \text{ bits/cell}
\]

Sum-rate without WOM = 3 bits/cell
Rewriting codes – Lattice-based WOM

\[ f(x, y) = 0 \]

\[ L_1 \]

\[ L_2 \]

Center for Magnetic Recording Research (CMRR)
Rewriting codes – Lattice-based WOM

\[ \text{Sum-rate} = |L_1| \cdot |L_2(x_1, y_1)| \]

\[ \text{Symmetric-rate} = 2 \cdot \min\{ |L_1|, |L_2(x_1, y_1)| \} \]
Rewriting codes – Lattice-based WOM

Sum-rate
\[ = |L_1| \cdot |L_2(x_2, y_2)| \]

Symmetric-rate
\[ = 2 \cdot \min\{|L_1|, |L_2(x_2, y_2)|\} \]
Rewriting codes – Lattice-based WOM

\[ \text{Sum-rate} = |L_1| \cdot |L_2(x_3,y_3)| \]

\[ \text{Symmetric-rate} = 2 \cdot \min\{ |L_1|, |L_2(x_3,y_3)| \} \]
**Theorem:** For both sum-rate optimal and symmetric-rate optimal, \( f^*(x, y) = (\ell_1 - x)(\ell_2 - y) - u \), where \( u \) satisfies the following:

- for sum-rate optimal
  \[ \frac{d(u\Gamma_2(u))}{du} = 0 \]
- for symmetric-rate optimal
  \[ \Gamma_2(u) = u \]
Rewriting codes – Lattice-based WOM

Generalization to more than 2 writes and more than 2 cells:

Sum-rate
\[ \sum \text{-rate} = |L_1| \cdot |L_2(x_1,y_1)| \cdot |L_3(x_2,y_2)| \]

Symmetric-rate
\[ \text{Symmetric-rate} = 3 \cdot \min \{ |L_1|, |L_2(x_1,y_1)|, |L_3(x_2,y_2)| \} \]
Theorem: Suppose the number of cells is $n$ and the number of writes is $t$. For both sum-rate optimal and symmetric-rate optimal case,

$$f_i(x) = \prod_{j=1}^{n} (\ell_j - x_j) - u_i, \quad i \in [1 : t - 1],$$

where $u_i$ satisfies the following:

Let $\Gamma_n(x) = 1 - x \sum_{i=0}^{n-1} (-1)^i \frac{(\ln x)^i}{i!}$, then

- for sum-rate optimal,
  $$d(u_1 \Gamma_n(u_1)) / du_1 = 0;$$
- for symmetric-rate optimal,
  $$\Gamma_n(u_1) = \Gamma_n(u_1') u_1,$$

where $u_1'$ is optimal parameter if the number of writes is $t - 1$. 

Center for Magnetic Recording Research (CMRR)
Rewriting codes – Lattice-based WOM

**TABLE I**

Worst-Case Sum-Rates $R_t$ in bits per cell per erase achieved by $t$-write codes on 2 cells with $q$ levels.

<table>
<thead>
<tr>
<th>$\log_2 q$</th>
<th>$t=2$</th>
<th>$t=3$</th>
<th>$t=3.59$</th>
<th>$t=4$</th>
<th>$t=5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q=2$</td>
<td>2.70</td>
<td>4.55</td>
<td>5.63</td>
<td>6.44</td>
<td>8.40</td>
</tr>
<tr>
<td>$q=3$</td>
<td>2.95</td>
<td>5.48</td>
<td>7.11</td>
<td>8.25</td>
<td>11.13</td>
</tr>
<tr>
<td>$q=4$</td>
<td>2.59</td>
<td>6.09</td>
<td>8.17</td>
<td>9.71</td>
<td>13.46</td>
</tr>
<tr>
<td>$q=5$</td>
<td>2.09</td>
<td>6.55</td>
<td>9.07</td>
<td>10.90</td>
<td>15.40</td>
</tr>
<tr>
<td>$q=6$</td>
<td>1.79</td>
<td>6.61</td>
<td>9.63</td>
<td>11.80</td>
<td>17.21</td>
</tr>
<tr>
<td>$q=7$</td>
<td>–</td>
<td>6.70</td>
<td>10.10</td>
<td>12.54</td>
<td>18.72</td>
</tr>
<tr>
<td>$q=8$</td>
<td>–</td>
<td>6.42</td>
<td>10.26</td>
<td>13.15</td>
<td>20.22</td>
</tr>
</tbody>
</table>
Outline

• Modulation codes
  • ICI-free codes for flash memories
  • Rewriting codes (write-once memories codes)
  • Constrained codes for phase-change memories

• Error correction codes
  • Erasure codes for distributed storage
  • Belief-propagation decoding of polar codes
    • Read/write strategy for flash memories
      • Read thresholds estimation
      • Parallel programming of flash memories
        • Discrete-level representation
        • Rank modulation
Constrained codes for PCM

• Cell states
  • Cell level is represented by the resistance
    • Amorphous/RESET state (0) and Crystalline/SET state (1)
    • Cell programming (changing states) is done by heating the cells
  • Heat accumulation
    • Avoid frequently programming a single cell
  • Cross-talk (ICI)
    • Avoid frequently programming adjacent cells
Outline

• Modulation codes
  • ICI-free codes for flash memories
  • Rewriting codes (write-once memories codes)
  • Constrained codes for phase-change memories

• Error correction codes
  • Erasure codes for distributed storage
  • Belief-propagation decoding of polar codes

• Read/write strategy for flash memories
  • Read thresholds estimation
  • Parallel programming of flash memories
  • Discrete-level representation
  • Rank modulation
Erasure codes for distributed storage
Erasure codes for distributed storage
Erasure codes for distributed storage

- Basic Schemes: Replication codes vs. Erasure codes

To tolerate 2 erasures

\[ \text{file} \rightarrow 1 \ 2 \ 3 \ 4 \ 1 \ 2 \ 3 \ 4 \ 1 \ 2 \ 3 \ 4 \]

\[ \text{file} \rightarrow 1 \ 2 \ 3 \ 4 \ \text{P1} \ \text{P2} \]
Erasure codes for distributed storage

• An \((n,k)\) Maximum Distance Separable (MDS) code provides a way to:

  • Take \(k\) packets and generate \(n\) packets \textit{of the same size} such that \textit{Any \(k\) out of \(n\) suffice to reconstruct the original \(k\) packets}

  * Optimal reliability for that given redundancy. Well-known and used frequently, e.g. Reed-Solomon (RS) codes

  * Each packet is stored at a different node, \textit{distributed} in a network.
Erasure codes for distributed storage

• An \((n,k)\) Maximum Distance Separable (MDS) code provides a way to:

• Take \(k\) packets and generate \(n\) packets of the same size such that Any \(k\) out of \(n\) suffice to reconstruct the original \(k\) packets

• Optimal reliability for that given redundancy. Well-known and used frequently, e.g. Reed-Solomon (RS) codes

• Each packet is stored at a different node, distributed in a network.
Erasure codes for distributed storage

- Current default Hadoop architecture

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

- 3x replication is HDFS current default.
- Very large storage overhead (3x).
- Very costly for BIG data
Erasure codes for distributed storage

• (14,10) RS codes used by Facebook

640 MB file => 10 blocks

• Small storage overhead (1.4x).
• Currently only 8% of Facebook’s data is Reed-Solomon encoded.
• Bottleneck: Repair problem
• Estimated if 50% of data is RS coded, the repair network traffic will saturate
Erasure codes for distributed storage

- We can tolerate $n - k = 4$ node failures
  - Most of the time we start with very few nodes in failure.
  - Read from any 10 nodes, send all data to 3’ who can repair the lost block.
  - High network traffic
  - High disk read
  - 10x more than the lost information
Erasure codes for distributed storage

- We can tolerate $n - k = 4$ node failures
- Most of the time we start with very few nodes in failure.
  - Read from any 10 nodes, send all data to 3' who can repair the lost block.
  - High network traffic
  - High disk read
  - 10x more than the lost information
Erasure codes for distributed storage

- We can tolerate \( n - k = 4 \) node failures
- **Most of the time** we start with very few nodes in failure.
- Read from any 10 nodes, send all data to 3’ who can repair the lost block.
  - High network traffic
  - High disk read
  - 10x more than the lost information

Center for Magnetic Recording Research (CMRR)
University of California, San Diego
Erasure codes for distributed storage

- We can tolerate \( n - k = 4 \) node failures
- **Most of the time** we start with very few nodes in failure.
- Read from any 10 nodes, send all data to 3’ who can repair the lost block.
- High network traffic
- High disk read
- 10x more than the lost information
Erasure codes for distributed storage

• Do I need to reconstruct the whole data object to repair very few failures?

*Figure of Merits:*

• The number of nodes accessed to repair a single node failure (Locality, locally repairable codes)
Locally repairable codes (LRC)

**Definition:**
- A code symbol has **locality** \( r \) if it is a function of \( r \) other codeword symbols.
- A systematic code has **message (information) locality** \( r \) if all its systematic symbols have locality \( r \).
- A code has **all-symbol locality** \( r \) if all its symbols have locality \( r \).

**Example:**
- In an MDS code, all symbols have locality \( r = k \).
- Only message locality is considered in practice (parities can be repaired offline).
Locally repairable codes (LRC)

**Theorem:**
For an \((n, k)\) codes with minimum distance \(d_{\text{min}}\), and all-symbol/message locality \(r\),
Singleton Bounds \((r = n - 1)\):
\[
d \leq n - k + 1 \quad \text{(bound by R. Singleton 1964)}
\]

All-symbol/message locality bounds:
\[
d \leq n - k - \left\lfloor \frac{k}{r} \right\rfloor + 2 \quad \text{(Gopalan et al.)}
\]
Locally repairable codes (LRC)

- Local XORs allow single block recovery by transferring only 5 blocks (320MB) instead of 10 blocks (640 MB in HDFS RAID).
- 17 total blocks stored
- Storage overhead increased to 1.7x from 1.4x
- Can be further reduced to 1.6x from 1.7x by IA
Locally repairable codes (LRC)

Extensions from LRC

1. Locality for multiple nodes in failure
   • Symbol-pair LRC

2. A single node failure requires multiple disjoint/non-overlapping reconstruction
   • Locality – availability trade-off
Locality for multiple nodes in failure

**Definition:**

- A pair of symbols of a code has locality \((L_1, L_2)\) if both symbols have locality \(L_1\), and the symbol-pair is a function of \(L_2\) other codeword symbols.
- A \((n, k, d_{\text{min}})\) code is an \((n, k, d_{\text{min}}, L_1, L_2)\) SP-LRC if it is of information SP locality \((L_1, L_2)\).

**Example:**

- In an MDS code, all symbols have SP locality \((L_1, L_2) = (k, k)\).
  - \((n, k)\) MDS code = \((n, k, n - k + 1, k, k)\) SP-LRC.
- In an \((n, k, r)\) LRC, it has SP locality \((L_1, L_2) = (r, k)\).
**Locality – availability trade-off**

**Definition:**
An \((n, k, d_{\text{min}})\) code is said to be an \((n, k, d_{\text{min}}, r, t)\)-local code if

- For each information (systematic) symbol \(c_i\),
  - \(t\) disjoint repair groups
  - size of each repair group at most \(r\).

**Remark:**
- Each systematic symbol has **locality** \(r\) and **availability** \(t\).
- \((r, 1)\)-local code = regular LRC with locality \(r\)
Locally repairable codes (LRC)

**Example:**
A (16, 9, 4) product code as a (3, 5) SP-LRC and a (3, 2)-local code

```
1  2  3  P_{123}
4  5  6  P_{456}
7  8  9  P_{789}
P_{147}  P_{258}  P_{369}  P_{\text{global}}
```
Locally repairable codes (LRC)

**Example:**
A (16, 9, 4) product code as a (3, 5) SP-LRC and a (3, 2)-local code

![Diagram of LRC product code example]
Locally repairable codes (LRC)

Example:
A (16, 9, 4) product code as a (3, 5) SP-LRC and a (3, 2)-local code

\[
\begin{array}{cccc}
1 & 2 & 3 & P_{123} \\
4 & 5 & \text{\(\star\)} & P_{456} \\
7 & 8 & 9 & P_{789} \\
\end{array}
\]

\[
\begin{array}{cccc}
P_{147} & P_{258} & P_{369} & P_{\text{global}} \\
\end{array}
\]
Locally repairable codes (LRC)

Example:
A (16, 9, 4) product code as a (3, 5) SP-LRC and a (3, 2)-local code
Locally repairable codes (LRC)

Example:
A (16, 9, 4) product code as a (3, 5) SP-LRC and a (3, 2)-local code
Locally repairable codes (LRC)

- 100 data nodes
- Some of them are erased
- Count how many nodes needs to be accessed for reconstruction
Outline

• Modulation codes
  • ICI-free codes for flash memories
  • Rewriting codes (write-once memories codes)
  • Constrained codes for phase-change memories

• Error correction codes
  • Erasure codes for distributed storage
  • Belief-propagation decoding of polar codes

  • Read/write strategy for flash memories
    • Read thresholds estimation
    • Parallel programming of flash memories
      • Discrete-level representation
      • Rank modulation
Belief-propagation decoding of polar codes

Standard decoding graph
Belief-propagation decoding of polar codes

Extended decoding graph

Standard decoding graph

\( U_{\text{bad}} \)

\( U_{\text{inter}} \)

\( U_{\text{good}} \)
Belief-propagation decoding of polar codes

- Improvements on soft-output decoder
  - Useful in inter-symbol interference channels (e.g., HDD)
Outline

• Modulation codes
  • ICI-free codes for flash memories
  • Rewriting codes (write-once memories codes)
  • Constrained codes for phase-change memories

• Error correction codes
  • Erasure codes for distributed storage
  • Belief-propagation decoding of polar codes

• Read/write strategy for flash memories
  • Read thresholds estimation
  • Parallel programming of flash memories
    • Discrete-level representation
    • Rank modulation
Read thresholds estimation

- Noise assumed Gaussian
- Adaptive thresholds to minimize the bit-error-rate
- Need to estimate the means and the variances from a minimum number of reads
Read thresholds estimation

- Noise assumed Gaussian
- Adaptive thresholds to minimize the bit-error-rate
- Need to estimate the means and the variances from a minimum number of reads

![Graph showing probability distributions with thresholds for different BER values.]

- $t_{\text{mean}}$ (BER = 5.6%)
- $t_{\text{median}}$ (BER = 4.8%)
- $t^*$ (BER = 4.5%)
Read thresholds estimation

- Noise assumed Gaussian
- Adaptive thresholds to minimize the bit-error-rate
- Need to estimate the means and the variances from a minimum number of reads

![Graph showing probability distribution with voltage and different thresholds]

- $t_{\text{mean}}$ (BER = 5.6%)
- $t_{\text{median}}$ (BER = 4.8%)
- $t^*$ (BER = 4.5%)
Outline

• Modulation codes
  • ICI-free codes for flash memories
  • Rewriting codes (write-once memories codes)
  • Constrained codes for phase-change memories

• Error correction codes
  • Erasure codes for distributed storage
  • Belief-propagation decoding of polar codes

• Read/write strategy for flash memories
  • Read thresholds estimation
  • Parallel programming of flash memories
    • Discrete-level representation
    • Rank modulation
Parallel programming for flash memories

- Find optimal voltage $\rightarrow$ Solve optimization problem
Outline

• Modulation codes
  • ICI-free codes for flash memories
  • Rewriting codes (write-once memories codes)
  • Constrained codes for phase-change memories
• Error correction codes
  • Erasure codes for distributed storage
  • Belief-propagation decoding of polar codes
• Read/write strategy for flash memories
  • Read thresholds estimation
  • Parallel programming of flash memories
    • Discrete-level representation
    • Rank modulation
Parallel programming for flash memories

Discrete-level representation

• \( \Theta = (\theta_1, \ldots, \theta_n) \): the target cell level vector
• \( \mathcal{V} = (V_1, V_2, \ldots, V_t) \): the voltages applied to the \( n \) cells during the \( t \) rounds of programming
• \( \ell_t = (\ell_{1,t}, \ldots, \ell_{n,t}) = f(\mathcal{V}, \varepsilon) \): the actual cell level vector after programming (assumed linear)
• Cost function \( C(\Theta, \ell_t) \): to evaluate the performance of the cell programming
• Goal: Given information of \( \Theta \), and programming noise \( \varepsilon \) (assumed additive Gaussian)

\[
\text{minimize } \mathbb{E}[C(\Theta, \ell_t)]
\]
with \( \mathcal{V} = (V_1, V_2, \ldots, V_t) \in \mathbb{R}^t_+ \)
Parallel programming for flash memories

Discrete-level representation

- \( \Theta = (\theta_1, ..., \theta_n) \): the target cell level vector
- \( V = (V_1, V_2, ..., V_t) \): the voltages applied to the \( n \) cells during the \( t \) rounds of programming
- \( \ell_t = (\ell_{1,t}, ..., \ell_{n,t}) = f(V, \varepsilon) \): the actual cell level vector after programming (assumed linear)
- Cost function \( C(\Theta, \ell_t) \): to evaluate the performance of the cell programming
- Goal: Given information of \( \Theta \) and programming noise \( \varepsilon \) (assumed additive Gaussian)

\[
\text{minimize } \mathbb{E}[C(\Theta, \ell_t)]
\]

with \( V = (V_1, V_2, ..., V_t) \in \mathbb{R}^t \)
Parallel programming for flash memories

Discrete-level representation

- $\Theta = (\theta_1, ..., \theta_n)$: the target cell level vector
- $V = (V_1, V_2, ..., V_t)$: the voltages applied to the $n$ cells during the $t$ rounds of programming
- $l_t = (l_{1,t}, ..., l_{n,t}) = f(V, \varepsilon)$: the actual cell level vector after programming (assumed linear)

- Cost function $C(\Theta, l_t)$: to evaluate the performance of the cell programming
- Goal: Given information of $\Theta$, and programming noise $\varepsilon$ (assumed additive Gaussian)

$$\min \mathbb{E}[C(\Theta, l_t)]$$

with $V = (V_1, V_2, ..., V_t) \in \mathbb{R}^t_+$
Parallel programming for flash memories

Discrete-level representation

- $\Theta = (\theta_1, \ldots, \theta_n)$: the target cell level vector
- $V = (V_1, V_2, \ldots, V_t)$: the voltages applied to the $n$ cells during the $t$ rounds of programming
- $l_t = (l_{1,t}, \ldots, l_{n,t}) = f(V, \epsilon)$: the actual cell level vector after programming (assumed linear)
- **Cost function** $C(\Theta, l_t)$: to evaluate the performance of the cell programming
- **Goal**: Given information of $\Theta$, and programming noise $\epsilon$ (assumed additive Gaussian)

$$\text{minimize } \mathbb{E}[C(\Theta, l_t)]$$

with $V = (V_1, V_2, \ldots, V_t) \in \mathbb{R}^t$
Parallel programming for flash memories

Discrete-level representation
• $\Theta = (\theta_1, ..., \theta_n)$: the target cell level vector
• $V = (V_1, V_2, ..., V_t)$: the voltages applied to the $n$ cells during the $t$ rounds of programming
• $l_t = (l_{1,t}, ..., l_{n,t}) = f(V, \varepsilon)$: the actual cell level vector after programming (assumed linear)
• Cost function $C(\Theta, l_t)$: to evaluate the performance of the cell programming
• Goal: Given information of $\Theta$, and programming noise $\varepsilon$ (assumed additive Gaussian)

$$\minimize \mathbb{E}[C(\Theta, l_t)]$$

with $V = (V_1, V_2, ..., V_t) \in \mathbb{R}^t_+$
Parallel programming for flash memories

**Result:**

- Solutions with polynomial-time complexity obtained if $\varepsilon = 0$
- Analytical solutions obtained if $n = 1$
Outline

• Modulation codes
  • ICI-free codes for flash memories
  • Rewriting codes (write-once memories codes)
  • Constrained codes for phase-change memories
• Error correction codes
  • Erasure codes for distributed storage
  • Belief-propagation decoding of polar codes
• Read/write strategy for flash memories
  • Read thresholds estimation
  • Parallel programming of flash memories
    • Discrete-level representation
    • Rank modulation
Parallel programming for flash memories

• Rank modulation: Discrete-levels vs. Permutations

Traditional representation

Level 1 2 0 2
Message = (1202) \in \{0,1,2,3\}^4
Number of messages = 4^4

Rank modulation

Level 1 2 0 3
Rank 2 3 1 4
Message = (2314) \in \Sigma_4
Number of messages = 4!
Parallel programming for flash memories

- Rank modulation: Discrete-levels vs. Permutations

Traditional representation

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>(1202)</td>
<td>∈</td>
<td>{0,1,2,3}^4</td>
<td></td>
</tr>
<tr>
<td>Number of messages</td>
<td>= 4^4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rank modulation

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rank</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Message</td>
<td>(2314)</td>
<td>∈</td>
<td>Σ_4</td>
<td></td>
</tr>
<tr>
<td>Number of messages</td>
<td>= 4!</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parallel programming for flash memories

Given:
- \( \ell_0 \) with permutation \( \text{permutation}(\ell_0) = \sigma \), (assume \( \sigma = (1,2,...,n) \) )
- Target permutation \( \tau \in \Sigma_n \)

Solve for:
- \( \mathbf{V} = (V_1, \ldots, V_t) \in \mathbb{Z}_+^t \)

Such that
- Let \( \ell_t = (\ell_{1,t}, \ldots, \ell_{n,t}) = f(\mathbf{V}) \): the actual cell level vector after programming (assumed linear)
- permutation(\( \ell_t \)) = \( \tau \)
- \( t \) is minimized, i.e., minimize the programming time
Parallel programming for flash memories

Given:
- \( \mathbf{l}_0 \) with permutation(\( \mathbf{l}_0 \)) = \( \sigma \), (assume \( \sigma = (1,2,\ldots,n) \))
- Target permutation \( \tau \in \Sigma_n \)

Solve for:
- \( \mathbf{V} = (V_1, \ldots, V_t) \in \mathbb{Z}^t_+ \)

Such that
- Let \( \mathbf{l}_t = (l_{1,t}, \ldots, l_{n,t}) = f(\mathbf{V}) \): the actual cell level vector after programming (assumed linear)
- permutation(\( \mathbf{l}_t \)) = \( \tau \)
- \( t \) is minimized, i.e., minimize the programming time
Parallel programming for flash memories

The diagram shows the average number of programming rounds required for different algorithms as a function of the number of cells. The algorithms compared are:

- **Optimal**
- **Algorithm 1 and 2**
- **Theorem 2**
- **No parallel**

The graph plots the average number of programming rounds against the number of cells, with the number of cells ranging from 4 to 10.
Summary

• Modulation codes
  • ICI-free codes for flash memories
  • Rewriting codes (write-once memories codes)
  • Constrained codes for phase-change memories
• Error correction codes
  • Erasure codes for distributed storage
  • Belief-propagation decoding of polar codes
• Read/write strategy for flash memories
  • Read thresholds estimation
  • Parallel programming of flash memories
    • Discrete-level representation
    • Rank modulation
Related papers

**Journal papers**


- Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, "**Optimized cell programming for flash memories with quantizers**", accepted by *IEEE Transaction on Information Theory*

- Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, “**Constrained codes that mitigate intercell interference in read/write cycles for flash memories**”, accepted by *IEEE Journal on Selected Areas in Communications*

- Aman Bhatia, Minghai Qin, Aravind Iyengar, Brian Kurkoski, and Paul H. Siegel, “**Lattice-based WOM codes for multilevel flash memories**”, accepted by *IEEE Journal on Selected Areas in Communications*
Related papers

Conferences and Workshops


Related papers

Conferences and Workshops

• Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, "Time-space constrained codes for phase-change memories", in Proc. IEEE Globecom 2011, Houston, Texas, USA, December 2011
• Borja Peleato, Rajiv Agarwal, John Cioffi, Minghai Qin and Paul H. Siegel, "Towards minimizing read time for NAND Flash", in Proc. IEEE Globecom 2012, Anaheim, CA, USA, December 2012
• Lele Wang, Minghai Qin, "Sum-capacity of multiple-write noisy memories", 2nd Non-volatile memories workshop, La Jolla, CA, USA, March 2011
• Minghai Qin, Lele Wang, Eitan Yaakobi, Young-Han Kim, and Paul H. Siegel, "WOM with history", 3rd Non-volatile memories workshop, La Jolla, CA, USA, March 2012
• Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, "Inter-cell Interference Free Codes for Read/write in Flash Memories", 4th Non-volatile memories workshop, La Jolla, CA, USA, March 2013
Acknowledgement

• Rajiv Agarwal
• Aman Bhatia
• John Cioffi
• Albert Guillén i Fàbregas
• Jing Guo
• Aravind Iyengar
• Anxiao Jiang
• Young-Han Kim
• Brian Kurkoski
• Borja Peleato
• Lele Wang
• Eitan Yaakobi