Fault-Tolerant Computing

Dealing with Mid-Level Impairments
About This Presentation

This presentation has been prepared for the graduate course ECE 257A (Fault-Tolerant Computing) by Behrooz Parhami, Professor of Electrical and Computer Engineering at University of California, Santa Barbara. The material contained herein can be used freely in classroom teaching or any other educational setting. Unauthorized uses are prohibited. © Behrooz Parhami

<table>
<thead>
<tr>
<th>Edition</th>
<th>Released</th>
<th>Revised</th>
</tr>
</thead>
</table>
Dear, your boss just called to tell you there was a slight mistake in your paycheck.

“We found the problem. You called your computer a moron and it’s waiting for an apology.”

“There’s nothing wrong with your iPod, Dad. It’s just too embarrassed to play the kind of music you like!”
Multilevel Model

Component

Logic

Information

System

Service

Result

Legend:

Ideal

Defective

Faulty

Erroneous

Malfunctioning

Degraded

Failed

Low-Level Impaired

Mid-Level Impaired

High-Level Impaired

Legend:

Ideal

Defective

Faulty

Erroneous

Malfunctioning

Degraded

Failed

Tolerance

Entry

Deviation

Remedy

Today

Last lecture

Oct. 2007
High-Redundancy Codes

Triplication is a form of error coding: 
x represented as xxx (200% redundancy)  
Corrects any error in one version  
Detects two nonsimultaneous errors

If we triplicate the voter to obtain 3 results, 
we are essentially performing the operation 
f(x) on coded inputs, getting coded outputs

With a larger replication factor, more errors can be tolerated

Our challenge today is to come up with strong correction capabilities, 
using much lower redundancy (perhaps an order of magnitude less)

To correct all single-bit errors in an n-bit code, we must have 2^r > n, 
or 2^r > k + r, which leads to about log_2 k check bits, at least
Error-Correcting Codes: Idea

A conceptually simple error-correcting code:
Arrange the $k$ data bits into a $k^{1/2} \times k^{1/2}$ square array
Attach an even parity bit to each row and column of the array
Row/Column check bit = XOR of all row/column data bits
Data space: All $2^k$ possible $k$-bit words
Redundancy: $2k^{1/2} + 1$ check bits for $k$ data bits
Corrects all single-bit errors (lead to distinct noncodewords)
Detects all double-bit errors (some triples go undetected)
Error-Correcting Codes: Evaluation

Redundancy: $k$ data bits encoded in $n = k + r$ bits ($r$ redundant bits)

Encoding: Complexity (cost / time) to form codeword from data word

Decoding: Complexity (cost / time) to obtain data word from codeword

Capability: Classes of error that can be corrected
Greater correction capability generally involves more redundancy
To correct $c$ bit-errors, a minimum code distance of $2c + 1$ is required

Examples of code correction capabilities:
Single, double, byte, $b$-bit burst, unidirectional, . . . errors

Combined error correction/detection capability:
To correct $c$ errors and additionally detect $d$ errors ($d > c$),
a minimum code distance of $c + d + 1$ is required

Example: Hamming SEC/DED code has a code distance of 4
Hamming Distance for Error Correction

The following visualization, though not completely accurate, is still useful

Red dots represent codewords

Yellow dots, noncodewords within distance 1 of codewords, represent correctable errors

Blue dot, within distance 2 of three different codewords represents a detectable error

Not all “double errors” are correctable, however, because there are points within distance 2 of some codewords that are also within distance 1 of another
A Hamming SEC Code

Uses multiple parity bits, each applied to a different subset of data bits

**Encoding:** 3 XOR networks to form parity bits

**Checking:** 3 XOR networks to verify parities

**Decoding:** Trivial (separable code)

**Redundancy:** 3 check bits for 4 data bits
Unimpressive, but gets better with more data bits
(7, 4); (15, 11); (31, 26); (63, 57); (127, 120)

**Capability:** Corrects any single-bit error

\[
\begin{align*}
  s_2 &= d_3 \oplus d_2 \oplus d_1 \oplus p_2 \\
  s_1 &= d_3 \oplus d_1 \oplus d_0 \oplus p_1 \\
  s_0 &= d_2 \oplus d_1 \oplus d_0 \oplus p_0
\end{align*}
\]

### Syndrome Table

<table>
<thead>
<tr>
<th>$s_2$</th>
<th>$s_1$</th>
<th>$s_0$</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$p_0$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$p_1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$d_0$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$p_2$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$d_2$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$d_3$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$d_1$</td>
</tr>
</tbody>
</table>
Matrix Formulation of Hamming SEC Code

\[
\begin{pmatrix}
d_3 & d_2 & d_1 & d_0 & p_2 & p_1 & p_0 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 \\
\end{pmatrix}
\times
\begin{pmatrix}
d_3 \\
d_2 \\
d_1 \\
d_0 \\
p_2 \\
p_1 \\
p_0 \\
\end{pmatrix}
= \begin{pmatrix}
s_2 \\
s_1 \\
s_0 \\
\end{pmatrix}
\]

Parity check matrix

Parity bits

Data bits

Matrix-vector multiplication is done with AND/XOR, instead of ×/+
Matrix Rearrangement for Simpler Correction

Data and parity bits
\[
\begin{pmatrix}
p_0 & p_1 & d_0 & p_2 & d_2 & d_3 & d_1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
p_0 \\
p_1 \\
d_0 \\
p_2 \\
d_2 \\
d_3 \\
d_1 \\
\end{pmatrix}
\times
\begin{pmatrix}
s_2 \\
s_1 \\
s_0 \\
\end{pmatrix}
= \begin{pmatrix}
s_2 \\
s_1 \\
s_0 \\
\end{pmatrix}
\]

Syndrome indicates error in position 4

Error
\[
\begin{array}{ccc}
s_2 & s_1 & s_0 \\
0 & 0 & 0 & \text{None} \\
0 & 0 & 1 & p_0 \\
0 & 1 & 0 & p_1 \\
0 & 1 & 1 & d_0 \\
1 & 0 & 0 & p_2 \\
1 & 0 & 1 & d_2 \\
1 & 1 & 0 & d_3 \\
1 & 1 & 1 & d_1 \\
\end{array}
\]
Hamming Generator Matrix

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 \\
\end{pmatrix}
\times
\begin{pmatrix}
d_3 \\
d_2 \\
d_1 \\
d_0 \\
\end{pmatrix}
=
\begin{pmatrix}
d_3 \\
d_2 \\
d_1 \\
d_0 \\
p_2 \\
p_1 \\
p_0 \\
\end{pmatrix}
\]

Recall that matrix-vector multiplication is done with AND/XOR, instead of \(\times/\)
Generalization to Wider Hamming SEC Codes

Data and parity bits
\[
\begin{array}{cccccc}
\ p_0 & p_1 & d_0 & p_2 & \ldots \\
\ 0 & 0 & 0 & \ldots & 1 & 1 & 1 \\
\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\ 0 & 1 & 1 & \ldots & 0 & 1 & 1 \\
\ 1 & 0 & 1 & \ldots & 1 & 0 & 1 \\
\end{array}
\]

\[= \begin{pmatrix} p_0 \\ p_1 \\ d_0 \\ p_2 \\ \vdots \\ s_{r-1} \\ s_1 \\ s_0 \end{pmatrix}
\]

Decoder

Data and parity bits

\[= \begin{pmatrix} 2^{r-1} \\ 2^{r-1} \end{pmatrix}
\]

Corrected version

\[= \begin{pmatrix} 2^{r-1} \\ 2^{r-1} \end{pmatrix}
\]

Condition for general Hamming SEC code:
\[n = k + r = 2^r - 1\]

<table>
<thead>
<tr>
<th>(n)</th>
<th>(k = n - r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>31</td>
<td>26</td>
</tr>
<tr>
<td>63</td>
<td>57</td>
</tr>
<tr>
<td>127</td>
<td>120</td>
</tr>
<tr>
<td>255</td>
<td>247</td>
</tr>
<tr>
<td>511</td>
<td>502</td>
</tr>
<tr>
<td>1023</td>
<td>1013</td>
</tr>
</tbody>
</table>
A Hamming SEC/DED Code

Data and parity bits

\[
p_0 \ p_1 \ d_0 \ p_2 \ldots
\]

\[
\begin{array}{cccc}
1 & 1 & 1 & \ldots & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & \ldots & 1 & 1 & 1 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
0 & 1 & 1 & \ldots & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & \ldots & 1 & 0 & 1 & 0 \\
\end{array}
\]

\[\begin{array}{c}
1 \\
2 \\
3 \\
\vdots \\
2^r-1
\end{array}\]

Position number

Parity check matrix

Add an extra row of all 1s and a column with only one 1 to the parity check matrix

\[
\begin{pmatrix}
\ p_0 \\
\ p_1 \\
\ d_0 \\
\ p_2 \\
\vdots \\
\ s_{r-1} \\
\ s_r
\end{pmatrix}
\times
\begin{pmatrix}
p_0 \\
p_1 \\
d_0 \\
p_2 \\
\vdots \\
s_{r-1} \\
s_r
\end{pmatrix}
= \begin{pmatrix}
s_r \\
s_{r-1} \\
s_1 \\
s_0
\end{pmatrix}
\]

Received word

Syndrome

Decoder

Corrected version
Some Other Useful Codes

Hamming codes are examples of linear codes
Linear codes may be defined in many other ways
There are also many other classes of codes

**BCH codes:** Named in honor of Bose, Chaudhuri, Hocquenghem

**Reed-Solomon codes:** Special case of BCH code
Example: A popular variant is RS(255, 223) with 8-bit symbols
223 bytes of data, 32 check bytes, redundancy $\approx 14\%$
Can correct errors in up to 16 bytes anywhere in the 255-byte codeword
Used in CD players, digital audio tape, digital television

**Reed-Muller codes:** Have a recursive construction, with smaller codes used to build larger ones

**Turbo codes:** Highly efficient separable codes, with iterative (soft) decoding
Reed-Solomon Codes

With \( k \) data symbols, require \( 2t \) check symbols, each \( s \) bits wide, to correct up to \( t \) symbol errors; hence, RS\((k + 2t, k)\) has distance \( 2t + 1 \)
The number \( k \) of data symbols must satisfy \( k \leq 2^s - 1 - 2t \) (\( s \) grows with \( k \))

<table>
<thead>
<tr>
<th>( k ) data symbols</th>
<th>( 2t ) check symbols</th>
</tr>
</thead>
</table>

Example: RS\((6, 2)\) code, with 2 data and \( 2t = 4 \) check symbols (7-valued)
→ up to \( t = 2 \) symbol errors correctable; hence, RS\((6, 2)\) has distance 5

Generator polynomial: \( g(x) = (x - \beta)(x - \beta^2)(x - \beta^3)(x - \beta^4) \);
\( \beta \) is a primitive root mod 7, that is, \( \beta^7 = 1 \) mod 7, but \( \beta^j \neq 1 \) mod 7 for \( j < 7 \)

Pick \( \beta = 3 \) → \( g(x) = (x - 3)(x - 3^2)(x - 3^3)(x - 3^4) \)
\[ = (x - 3)(x - 2)(x - 6)(x - 4) = x^4 + 6x^3 + 3x^2 + 2x + 4 \]

As usual, the codeword is the product of \( g(x) \) and the info polynomial; convertible to matrix-by-vector multiply by deriving a generator matrix \( G \)
BCH Codes

Correct the deficiency of Reed-Solomon code; have a fixed alphabet
We usually choose the alphabet \{0, 1\}

BCH(15, 7) code: Capable of correcting any two errors
Generator polynomial: \( g(x) = 1 + x^4 + x^6 + x^7 + x^8 \)

\[
\begin{bmatrix}
0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix} \times \\
\begin{bmatrix}
\end{bmatrix} = [x \ x \ x \ x \ x \ x \ x]
\]

Received word
Parity check matrix
Syndrome
### Arithmetic Error-Correcting Codes

<table>
<thead>
<tr>
<th>Positive error mod 7</th>
<th>Syndrome mod 15</th>
<th>Negative error</th>
<th>Syndrome mod 7</th>
<th>Syndrome mod 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-2</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>-4</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>-8</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>-16</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>-32</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>64</td>
<td>1</td>
<td>-64</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>128</td>
<td>2</td>
<td>-128</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>256</td>
<td>4</td>
<td>-256</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>512</td>
<td>2</td>
<td>-512</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>1024</td>
<td>2</td>
<td>-1024</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>2048</td>
<td>4</td>
<td>-2048</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4096</td>
<td>1</td>
<td>-4096</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>8192</td>
<td>2</td>
<td>-8192</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>16,384</td>
<td>4</td>
<td>-16,384</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>32,768</td>
<td>1</td>
<td>-32,768</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Error syndromes for weight-1 arithmetic errors in the (7, 15) biresidue code.

Because all the syndromes in this table are different, any weight-1 arithmetic error is correctable by the (mod 7, mod 15) biresidue code.
Properties of Biresidue Codes

Biresidue code with relatively prime low-cost check moduli $A = 2^a - 1$ and $B = 2^b - 1$ supports $a \times b$ bits of data for weight-1 error correction.

Representational redundancy $= (a + b)/(ab) = 1/a + 1/b$

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$n=k+a+b$</th>
<th>$k=ab$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>19</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>41</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>71</td>
<td>56</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>143</td>
<td>120</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>271</td>
<td>240</td>
</tr>
</tbody>
</table>

Compare with Hamming SEC code

<table>
<thead>
<tr>
<th>$n$</th>
<th>$k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>31</td>
<td>26</td>
</tr>
<tr>
<td>63</td>
<td>57</td>
</tr>
<tr>
<td>127</td>
<td>120</td>
</tr>
<tr>
<td>255</td>
<td>247</td>
</tr>
<tr>
<td>511</td>
<td>502</td>
</tr>
<tr>
<td>1023</td>
<td>1013</td>
</tr>
</tbody>
</table>

Oct. 2007  
Error Correction  
Slide 20
Arithmetic on Biresidue-Coded Operands

Similar to residue-checked arithmetic for addition and multiplication, except that two residues are involved.

Divide/square-root: remains difficult
Higher-Level Error Coding Methods

We have applied coding to data at the bit-string or word level

It is also possible to apply coding at higher levels

Data structure level – Robust data structures
Application level – Algorithm-based error tolerance
Preview of Algorithm-Based Error Tolerance

Error coding applied to data structures, rather than at the level of atomic data elements.

Example: mod-8 checksums used for matrices

If \( Z = X \times Y \) then \( Z_f = X_c \times Y_r \)

In \( M_f \), any single error is correctable and any 3 errors are detectable

Four errors may go undetected

\[
M = \begin{pmatrix}
2 & 1 & 6 \\
5 & 3 & 4 \\
3 & 2 & 7 \\
\end{pmatrix}
\]

\[
Mr = \begin{pmatrix}
2 & 1 & 6 & 1 \\
5 & 3 & 4 & 4 \\
3 & 2 & 7 & 4 \\
\end{pmatrix}
\]

\[
Mf = \begin{pmatrix}
2 & 1 & 6 & 1 \\
5 & 3 & 4 & 4 \\
3 & 2 & 7 & 4 \\
2 & 6 & 1 & 1 \\
\end{pmatrix}
\]

\[
M = \begin{pmatrix}
2 & 1 & 6 \\
5 & 3 & 4 \\
3 & 2 & 7 \\
\end{pmatrix}
\]

\[
Mf = \begin{pmatrix}
2 & 1 & 6 & 1 \\
5 & 3 & 4 & 4 \\
3 & 2 & 7 & 4 \\
2 & 6 & 1 & 1 \\
\end{pmatrix}
\]