Software Diversity for Hardware Fault Detection

Center for Reliable Computing
Software Diversity for Hardware Fault Detection
Nahmsuk Oh
Center for Reliable Computing
Stanford University
RATS Seminar
Jan 10, 2000

Outline

- Introduction
- Determination of $k$
  - Bus signal line
  - Adder
- Simulation Result
- Benchmark Simulation
- Conclusion

Software Diversity

- Previous approach
  - Target: Software design faults
  - (ex) N version programming
- Our approach
  - Target: Hardware faults
  - Original & Diverse program
    - Use different part of hardware

Software Diversity

- In a Diverse Program
  - All variables & constants
    - Multiplied by $k$
  - Bits might be reversed in:
    - Registers, memory
    - Data path, ALU, multiplier

Transformation

- Transformation to Diverse Program
  - Expression transformation
    - Variables multiplied by $k$
  - Condition transformation
    - Control flow preserved

Diversity Metric $d$

- Quantify the diversity[Mitra99]
  - Transformed program
- $d$: diversity metric
  - $N$: # of vectors applied
  - $m$: # of vectors
    - In original & diverse programs
    - generate the same incorrect outputs
  
$d = (N - m) / N$
Software Diversity for Hardware Fault Detection

## Determination of $k$

- $k$ should maximize:
  - Diversity metric $d$
  - $Pr$\{detecting faults\}
- $k$ should not cause:
  - Overflow in functional units

### Bus Signal Lines ($k=2$)

#### Original Program

<table>
<thead>
<tr>
<th>0 0 0 1 1 0 1 0</th>
</tr>
</thead>
</table>

#### Diverse Program

| 0 0 1 1 1 1 0 0 |

$0 0 1 0 1 0 \rightarrow x = 0 0 0 1 1 1 0$

- Assuming random inputs & no overflow
  - $d = 1$
  - $Pr$\{detecting fault\} = 3/4

### Bus Signal Lines ($k=-2$)

#### First 1

<table>
<thead>
<tr>
<th>(00010100100)</th>
<th>(x)</th>
<th>(-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1 0 0 0 0 0</td>
<td>(x)</td>
<td>(-1)</td>
</tr>
</tbody>
</table>

Possible 4 combinations of $x$ with equal probability:

<table>
<thead>
<tr>
<th>(x)</th>
<th>(-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1 0 0 0 0 0</td>
<td>(-1)</td>
</tr>
</tbody>
</table>

#### First 2

<table>
<thead>
<tr>
<th>(00010100100)</th>
<th>(x)</th>
<th>(-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1 0 0 0 0 0</td>
<td>(x)</td>
<td>(-1)</td>
</tr>
</tbody>
</table>

Possible 4 combinations of $x$ with equal probability:

<table>
<thead>
<tr>
<th>(x)</th>
<th>(-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1 0 0 0 0 0</td>
<td>(-1)</td>
</tr>
</tbody>
</table>

#### First 3

<table>
<thead>
<tr>
<th>(00010100100)</th>
<th>(x)</th>
<th>(-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1 0 0 0 0 0</td>
<td>(x)</td>
<td>(-1)</td>
</tr>
</tbody>
</table>

Possible 4 combinations of $x$ with equal probability:

<table>
<thead>
<tr>
<th>(x)</th>
<th>(-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1 0 0 0 0 0</td>
<td>(-1)</td>
</tr>
</tbody>
</table>

### Ripple Carry Adder

### Detection Probability

- Assumption
  - Equal prob. in all possible inputs
- Parker&McCluskey [Parker75]
  - Calculate prob. of logic circuit being 1
- Detection prob. includes:
  - Invoke the fault
  - Sensitize the path to the output
Software Diversity for Hardware Fault Detection

Simulation

- Apply:
  - All possible input combinations
- 12B RC adder

Simulation

- 12B CLA adder

Benchmark Simulation

- Integer benchmark programs
  - Running in MIPS simulator
  - Inject a fault into the adder
    - 32B RC adder
    - 32B CLA adder

7 Benchmark Simulation
- RC Adder

7 Benchmark Simulation
- CLA Adder

Conclusion

- Program with only arithmetic operations
  - As long as $k$ causes no overflow:
    - If $k > 0$, greater $k$ better
    - For the same $|k|$, negative $k$ better
    - But, high cost in negating number
- Programs with arithmetic and logical op.
  - Only $k = 2^l$ possible
  - Shifting $l$ bits
Reference