Control Flow Error Detection

A collection of techniques

Philip P. Shirvani
RATS
Center for Reliable Computing
Stanford University

May 8, 1995

Outline

- Introduction
- Classification
- Error detection
- Control flow error detection
- Signature analysis
- Execution time and address information
- Comparison
- Summary
- Research directions

Introduction

- Fault-tolerant systems are necessary for critical applications
- Reliability, e.g., avionics, railway control systems
- Not accessible for repair, e.g., satellite, space ships
- Fault avoidance
- High-quality components, conservative design, etc.
- Fault detection
- Fault tolerance
  - Permanent faults: reconfigure
  - Transient faults: undo and restart

Classification of Error Detection Methods

- Circuit level (hardware)
  - Parity, e.g., bus
  - SEC/DED, e.g., memory
  - Residue codes, e.g., ALU
  - Self-checking circuits
- System level (software and/or hardware)
  - Watchdog timer
  - Capability-based addressing
  - Duplication
  - N-version programming
  - Watchdog processor

Error Detection

- Information gathered during compile-time compared with information gathered during run-time
- What to check?
  - Control flow
  - Control signals
  - Memory access behavior
  - Assertion (reasonableness of results)
- How to check?
  - Self-checking
  - Separate task
  - Watchdog processor

Control Flow Error Detection

- Check the correct sequencing of the instructions
- Checking capability
  - Blocks are executed in an allowed sequence (interblock)
  - Sequencing of the contents of a block (intrablock)
  - Both
Control Flow Error Detection

Watchdog Processor
A small and simple processor that detects errors by monitoring the behavior of a system.

Signature Analysis
- assigned signature
  - watchdog program generated directly from HLL
  - two processors operate asynchronously without making the watchdog complex
  - signatures have to be explicitly transferred to watchdog
  - only interblock checking

E.g.: Structural Integrity Checking (SIC)
- HLL control flow structures
- structural reference program for watchdog:
  - 'computations' replaced by 'receive and check labels'

Signature Analysis
- derived signature
  - higher coverage
  - can do both inter- and intrablock checking
  - used in most signature based methods
  - assume program is not run-time modifiable
  - semantics or correct execution of instructions can not be directly checked

Two Pure Software Methods
- Block Signature Self-Checking (BSSC)
  - entry CALL -> save address of 'instruction 1'
  - instruction 1
  - instruction 2
  - ...
  - instruction n
  - exit CALL
  - embedded sign.
  - -> compare with embedded signature
  - -> address of 'instruction 1'

- Error Capturing Instructions (ECI)
  - ECIs in unused memory: trap error and recover
  - can be added to simple systems with watchdog timer

Signatured Instruction Streams (SIS)
- basic block:
  - a branch-free sequence of instructions
- intermediate signature:
  \[ S_k = V( S_{k-1}, W_{k-1}) \]

Philip - RATS Center for Reliable Computing May 8, 1995
### Branch Address Hashing (BAH)

- Combine signature with branch target address (xor)
- Embed signatures before branch-in points (merge points)
- Error detection at the next merge point
- Program bound detector

- Number of signatures reduced by as much as 50%
- Incorrect branch to a merge point not detected

### Path Signature Analysis (PSA)

**NOP for CPU**

<table>
<thead>
<tr>
<th>Signature</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Signature</td>
</tr>
<tr>
<td>00</td>
<td>1st instruction</td>
</tr>
<tr>
<td>00</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td></td>
</tr>
</tbody>
</table>

- Signals time to compare

#### Reducing Memory Overhead in PSA

- Signatures for sequences of blocks, i.e., paths rather than single blocks
- Each path set (paths starting from the same block) has one signature
- Needs justifying signatures

### Signature Embedding Implementation

- Pseudo-branch instruction (psbr)
  - Use prefetch of CPU, e.g., MC68000

#### Implementation of BAH

- Memory
- XOR
- LFSR
- Decode branch
- Main processor
Control Flow Error Detection

Asynchronous SIS (ASIS)

- control flow information as signature graph in local memory of RMP (Roving Monitoring Processor)
- RMP fetches signature and compares with the signature of all the possible succeeding nodes of the current node
- interrupts: signature stack in SG and RMP
- no performance overhead

Continuous Signature Monitoring (CSM)
- correlation among intermediate signatures decreases error coverage:
  - initial signature (S₀)
  - common language constructs
- uncorrelate:
  - program graph -> maximal paths
  - signature at entry and exit = 0
  - use intermediate signature at branch as S₀ of the new path
  - embed justifying signatures (like PSA)
- result: same signature produced along any route from entry to exit = one check point (increases latency)

Reducing Signature Cost
- cost: a function of memory overhead and performance loss
- signature placement:
  - can replace NOPs used to resolve pipeline interlocks
  - branch delay slots
  - placed at locations that are executed less frequently
- partition the graph into a maximal path set with the minimum total signature cost

Reducing Error-detection Latency
- use horizontal vs. vertical signatures
  - one cycle latency
  - no performance loss
  - less coverage with constant memory overhead
- two-dimensional signatures
  - horizontal: low latency; vertical: high coverage

CSM - “Continuous” Signature Monitoring
- horizontal signatures are checked continuously
- justifying signatures maintain vertical continuity between maximal paths
### CSM with Parity and SEC/DED
- Combining CSM with parity: hashed parity bit = $\overline{IP}$
- Can also be combined with SEC/DED
  - Reduces latency to a theoretical value of .016 mem. cy.
  - With this combination, transient monitor errors are detected, so there is no need to duplicate the monitor.
  - With this low latency, in pipeline processors, transient fault tolerance can be done by squashing the pipeline and restarting = no need for rollback/recovery buffers.

### CSM + Encryption or Instruction Hashing
- Generalization of BAH: hash each word using intermediate signature of that location (no multiplexer faster)
- Error results in a pseudo-random instruction
  - Trigger numerous error detection mechanisms
  - Reduce latency
  - Error captured by other mechanisms if monitor fails to detect or report
- Add a small cache to the monitor; instead of justifying signatures, look up intermediate signature of destination address of branch
  - Reduces CSM memory overhead.

### Instruction Hashing
- Computer virus resistance: use a cryptographic function for signature (must execute in real-time)
- Monitor assisted signature compilation: compiler gives each piece of code to the monitor and gets the signature
  - Including internal control signals becomes feasible
  - Higher coverage
  - Significant additional virus resistance

### Extended-Precision Checksums
- Instructions are space compacted
- Addition is performed without any loss of precision

<table>
<thead>
<tr>
<th>Transmit checksum</th>
<th>→ send checksum to the watchdog</th>
</tr>
</thead>
<tbody>
<tr>
<td>instruction 1</td>
<td>→ subtract instruction from checksum</td>
</tr>
<tr>
<td>instruction 2</td>
<td>→ subtract from previous result</td>
</tr>
<tr>
<td></td>
<td>→ subtract</td>
</tr>
<tr>
<td></td>
<td>→ watchdog checks for zero result</td>
</tr>
</tbody>
</table>

### Extended-Precision Checksums (continued)
- Error if:
  - Nonzero result when checked
  - Negative result before zero check signal/lower latency
  - Zero check signal before checksum transmitted
- The lower the checksum value the lower the latency: encode opcodes in respect to frequency
- Sequence error:
  - Coverage approaches unity vs. remaining constant
  - Latency remains bounded by the average block length vs. increasing

### On-line Signature Learning and Checking (OSLC)
- General configuration like ASIS
  - No modification of compiler/assembler: block identification and reference signature generation are done in a normal execution of program called learning phase; signatures are calculated by hardware and stored somewhere
  - No need for control flow graph
    - Program divided into sections with N signatures in each
    - Signatures stored in segments in memory of the monitor
    - SG sends signature and address of last instruction
    - No error if signature found in segment corresponding to address
Control Flow Error Detection

OSLC (continued)
- interrupts:
  - stacks in SG, but no stack in monitor; or
  - delay INT acceptance until current block reaches its end
- can include operand size, address and control signals
  - detect more types of errors
  - uncorrelate signatures
  - high coverage
- can not use justification signature when signature in local memory of monitor

Execution Time and Address Methods
- Basic Block (BB) and Partition Block (PB)
  - extra instructions added to the beginning and end of BB
  - partition blocks: instruction blocks between BBs
- Time-Time-Address (TTA) and Signature-Time-Address (STA)

Mechanisms Included in the Schemes

<table>
<thead>
<tr>
<th>Schemes</th>
<th>BB-timer</th>
<th>PB-timer</th>
<th>WL-timer</th>
<th>BB-address</th>
<th>Phase</th>
<th>CRC</th>
<th>BB-OP-counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>STA</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

TTA and STA
- BB-timer: exact execution time (clock or memory cycle)
- PB-timer: upper bound for execution time
- WL-timer: a traditional watchdog timer for monitoring workload
- BB-address: exit address = start address + size of block
- Phase: entering/exiting of the BBs occur in the correct order
- CRC: BB instruction signature
- BB-OP-counter: number of instructions executed
- TTA: can be adapted for external monitoring of processors with internal caches and nondeterministic execution time
- STA: can not be used for external monitoring of processors with internal caches

Comparison

<table>
<thead>
<tr>
<th>Method</th>
<th>Control flow coverage</th>
<th>Overall coverage</th>
<th>Latency (cycles)</th>
<th>Memory overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSSC+ECI</td>
<td>97%</td>
<td>73%</td>
<td>31-40%</td>
<td></td>
</tr>
<tr>
<td>Basic</td>
<td>96-99%</td>
<td>2.5</td>
<td>10-25%</td>
<td></td>
</tr>
<tr>
<td>PSA</td>
<td>99.5-99.9%</td>
<td>7-11</td>
<td>12-21%</td>
<td></td>
</tr>
<tr>
<td>SIS+BAH</td>
<td>88-93%</td>
<td>7-17</td>
<td>6-13%</td>
<td></td>
</tr>
<tr>
<td>CSM</td>
<td>99.9999%</td>
<td>0.016-1</td>
<td>4-11%</td>
<td></td>
</tr>
<tr>
<td>OSLC</td>
<td>99.4%</td>
<td>94.5%</td>
<td>(25.4s)</td>
<td>0%</td>
</tr>
<tr>
<td>TTA</td>
<td>98%</td>
<td>87-93%</td>
<td>77±</td>
<td>35-39%</td>
</tr>
<tr>
<td>STA</td>
<td>98%</td>
<td>88%</td>
<td>101±</td>
<td>35-39%</td>
</tr>
</tbody>
</table>

- coverages are affected by workload and fault-injection mode
  1 with watchdog timer
  2 overall latency

Summary
- use the capabilities of advanced compilers
- we have good low-cost techniques with high coverage and low latency for control flow errors
- can be implemented to new processors
- combine with other techniques that detect other type of errors not covered by above

Research Directions
- type of faults and their frequency
- occurrence of common-mode failure
- other error detection techniques
- effectiveness of various fault-tolerance techniques