Control Flow Checking by Instructions

Motivation
- Error detection capability:
  - added to general microprocessors
  - can be used for fault tolerance
  - generated automatically

Goal
- CAP: Control Flow Analysis Producer
  - Input: C source code
  - Output: executable with control flow checking
- Fully automatic
- Generated executable file
  - detect error during run-time
  - report error

Preliminary
- Basic block
- Control Flow Graph

Control Flow Checking
- Control flow checking in
  - Intra-block
  - Inter-block

Inter-Block CFC
- Signature analysis by instruction
  During compile:
  1. Assign signature number \( s \) to all BB
  2. Embed xor-difference between two nodes
     (xor-difference: \( d \))
  During run-time:
  3. Global signature register \( G \)
     At each node, update \( G = G \ xor \ d \) (=\( s \))
  4. Compare \( s \) & \( G \)
Control Flow Checking by Instructions

Inter-Block CFC
- Signature analysis by instruction

Control Flow Error Detection
- Illegal branch to checking instruction

Control Flow Error Detection
- Illegal branch to normal instruction

Calling Subroutine
- Treat subroutine as an independent CFG. Initialize G at the start node.

Intra-block CFC
- Intra-block control flow checking
  - Duplicated Instructions
  - Primary & secondary registers (memory)
  - Compare primary & secondary

Duplication
- Registers (total 72 reg)
  - A half for primary registers (36 reg)
  - A half for secondary registers (36 reg)
- Stack
  - Double the stack size
  - Duplication of local variables
Control Flow Checking by Instructions

**Duplication Example**
- **Source code**
  ```c
  Main()
  {
    int a, b, c;
    c = a + b;
  }
  ```
- **Manually written duplication**
  ```c
  Main()
  {
    int a, b, c;
    int a1, b1, c1;
    c = a + b;
    c1 = a1 + b1;
    if (c != c1) error;
  }
  ```

**Example (assembly code)**
- **Source code**
  ```c
  Main()
  {
    int a, b, c;
    c = a + b;
  }
  ```
- **Assembly code**
  ```
  R1 <- mem(sp + 1)
  R2 <- mem(sp + 2)
  R3 <- R1 + R2
  mem(sp + 3) <- R3
  ```

**Example (memory map)**
- **Assembly code**
  ```
  R1 <- mem(sp + 1)
  R2 <- mem(sp + 2)
  R3 <- R1 + R2
  mem(sp + 3) <- R3
  ```

**Duplication Example**
- **Assembly code**
  ```
  R1 <- mem(sp + 1)
  R2 <- mem(sp + 2)
  R3 <- R1 + R2
  R37 <- mem(sp + 11)
  R38 <- mem(sp + 12)
  R39 <- R37 + R38
  bne R3 != R39 , error
  mem(sp + 3) <- R3
  mem(sp+39) <- R39
  ```

**CAP**
- **Control flow Analysis Producer**
- **Target fault**
  - Bit flip in the memory
  - Transient error in hardware
  - Multiple faults easier to detect

**CAP Structure**

---

Nahmsuk Oh  
Page 3  
11/12/98
Control Flow Checking by Instructions

Overhead of SAI

<table>
<thead>
<tr>
<th>Program</th>
<th>Instructions</th>
<th>Checking Inst</th>
<th>Basic Mode</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT</td>
<td>471</td>
<td>100</td>
<td>47</td>
<td>34 %</td>
</tr>
<tr>
<td>adder_check</td>
<td>426</td>
<td>138</td>
<td>66</td>
<td>32 %</td>
</tr>
<tr>
<td>hash_table</td>
<td>1149</td>
<td>376</td>
<td>199</td>
<td>32 %</td>
</tr>
<tr>
<td>binary_tree</td>
<td>3330</td>
<td>1643</td>
<td>559</td>
<td>49 %</td>
</tr>
<tr>
<td>external_sort</td>
<td>1430</td>
<td>776</td>
<td>280</td>
<td>34 %</td>
</tr>
<tr>
<td>symbol_table</td>
<td>303</td>
<td>224</td>
<td>91</td>
<td>36 %</td>
</tr>
<tr>
<td>quick_sort</td>
<td>334</td>
<td>142</td>
<td>47</td>
<td>56 %</td>
</tr>
<tr>
<td>counter</td>
<td>46</td>
<td>19</td>
<td>9</td>
<td>47 %</td>
</tr>
<tr>
<td>lzw</td>
<td>452</td>
<td>280</td>
<td>105</td>
<td>61 %</td>
</tr>
<tr>
<td>insert_sort</td>
<td>192</td>
<td>84</td>
<td>29</td>
<td>64 %</td>
</tr>
<tr>
<td>simple_sort</td>
<td>191</td>
<td>130</td>
<td>41</td>
<td>68 %</td>
</tr>
</tbody>
</table>

CAP

- Status
  - Inter-block CFC implemented (90%)
  - Currently implementing Intra-block CFC

- Future work
  - Implement fault injection program
  - Simulate
  - Obtain fault coverage
  - Assess size and performance overhead

Summary

- Pros
  - Error detection capability for general microprocessor
  - Automatic generation
  - Can be used for fault tolerance

- Cons
  - Size overhead
  - Performance loss