Argus: Low-Cost, Comprehensive Error Detection in Simple Cores

Yilei Xu, Jiong Xue

9/30/2015
Outline

• Motivation
• Argus Overview
• Argus-1 Implementation
• Evaluation
• Conclusion
Motivation

• Why simple core?

• What’s important?

• Why Argus?
  • Dual Modular Redundancy
  • DIVA
  • Redundant Multithreading
  • ...

Argus Overview

• Low-cost

• Low-power

• Comprehensively detecting transient and permanent errors

• Run-time checking

• Checkpoint Recovery
Argus Overview

• Control Flow Checking
• Dataflow Checking
• Computation Checking
• Memory Checking
Implementation

Baseline OpenRISC processor

- 32-bit 1 way scalar
- In-order
- 4-stage pipeline
- 32 general purpose registers
- Instruction cache and data cache
Implementation
Implementation

State History Signature (SHS)

• Represents a creation history of current location’s state

• Associated with each architectural location
  • $\text{SHS}_{\text{reg}}$, $\text{SHS}_{\text{pc}}$, $\text{SHS}_{\text{mem}}$

• Depends on previous data and operation’s history
  • Add r1, r2, r3
Implementation

Dataflow and Control Signature (DCS)

• $F(\text{SHSs})$ for a block

• Static DCS
  • Computed at compile time
  • Embedded in unused instruction bits

• Dynamic DCS
  • Computed at run time
Implementation

Control Flow and Dataflow Checkers

Static DCS

BB1

BB2

BB3

BB4

Dynamic DCS

BB1

BB2

BB3

Error!

Control flow

Data flow
Implementation

Computation Checker

• Several functional unit sub-checkers
• ALU Sub-Checker
• Multiplier/Divider Sub-Checker
Implementation

ALU Sub-Checker
Implementation

Multiplier/Divider Sub-Checker
Implementation

Memory Checker

• Parity bits for D$ and MEM

• $D_A = D \text{ XOR } A$

• No memory order violations check
Experimental Evaluation

Error Detection Coverage

- Uncovered Errors
  - Memory ordering violations
  - DCS aliasing
  - Multiplication aliasing

<table>
<thead>
<tr>
<th>Error Type</th>
<th>unmasked, undetected</th>
<th>unmasked, detected</th>
<th>masked, undetected</th>
<th>masked, detected (DME)</th>
</tr>
</thead>
<tbody>
<tr>
<td>transient</td>
<td>0.76%</td>
<td>37.4%</td>
<td>38.2%</td>
<td>23.7%</td>
</tr>
<tr>
<td>permanent</td>
<td>0.46%</td>
<td>37.6%</td>
<td>38.2%</td>
<td>23.7%</td>
</tr>
</tbody>
</table>
Experimental Evaluation

Error Detection Latency

• Control/Data flow error - one basic block
• Computational error - one cycle
• Memory error - around one cycle
Experimental Evaluation

Area Overhead

<table>
<thead>
<tr>
<th></th>
<th>OR1200</th>
<th>With Argus-1</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>core</td>
<td>6.58</td>
<td>7.67</td>
<td>16.6%</td>
</tr>
<tr>
<td>I-cache: 1-way</td>
<td>2.14</td>
<td>2.14</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>2.42</td>
<td>2.42</td>
<td></td>
</tr>
<tr>
<td>D-cache: 1-way</td>
<td>2.14</td>
<td>2.24</td>
<td>4.9%</td>
</tr>
<tr>
<td></td>
<td>2.42</td>
<td>2.54</td>
<td>5.1%</td>
</tr>
<tr>
<td>total: 1-way</td>
<td>10.86</td>
<td>12.05</td>
<td>10.9%</td>
</tr>
<tr>
<td></td>
<td>11.42</td>
<td>2.63</td>
<td>10.6%</td>
</tr>
</tbody>
</table>

TABLE 2. Area Overhead. Areas in mm².
Experimental Evaluation

Performance Overhead

Figure 5. Dynamic Instruction Overhead
Conclusion

• Invariant checking methodology
• Low cost in area and performance
• Possible improvements
Discussion Points

• Can Argus check design error?

• What cannot be checked?

• Can Argus be used in complex cores?
Thank you!