Dynamic Derivation and Hardware Implementation of Error-Detectors
Karthik Pattabiraman, Giacinto Paolo Saggese, Zbigniew Kalbarczyk, Ravishankar K. Iyer

Research Goal
Application-aware error detectors
- Provide application-specific error detection at low-cost for embedded and high-performance platforms
- Limit error propagation to ensure crash-failure semantics
- Automatically derive fine-grained detectors to
  - Maximize error detection coverage
  - Minimize performance impact
- Implement in hardware / software:
  - Static and dynamic reconfigurability
  - Reliability and Security Engine (RSE)

Detector Placement
Choose both variable and location to place the detector
Construction of program’s Dynamic Dependence Graph (DDG), based on code and execution
Computed metrics (heuristics) to choose candidate points for placement without fault-injections
E.g., Fanout, Lifetime, Execution, Propagation and Coverage
Evaluation of detectors placed accordingly to different metrics
Fault-injections into data
Offline comparison with golden run (ideal detectors)

- Fanouts, Propagation metrics have best overall coverage
  - Crash coverage: 90% coverage for Siemens Suite, 60% for gcc, perl
  - Fail-Silent Violations coverage: 30 to 60% coverage for both application suites
  - False Positives rate: less than 5% for both application suites

Detector Classes
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Example of Checking Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>a[i] == c</td>
</tr>
<tr>
<td>Alternate</td>
<td>(a[i] == y and a[i-1] == y) or (a[i] == y and a[i-1] == x)</td>
</tr>
<tr>
<td>Multi-Value</td>
<td>a[i] in Values, where Values is a set of possible values</td>
</tr>
<tr>
<td>Range</td>
<td>min &lt;= a[i] &lt;= max</td>
</tr>
<tr>
<td>ConstantDiff</td>
<td>(a[i] – a[i-1]) == c</td>
</tr>
<tr>
<td>BoundedDiff</td>
<td>min &lt;= (a[i] – a[i-1]) &lt;= max</td>
</tr>
</tbody>
</table>

Process Workflow
1. Profiling program to build the Dynamic Dependence Graph (DDG)
2. Placement of detectors in the code to maximize coverage
3. Automated hardware generation to implement selected detectors
4. Instrumentation of code to interface with the custom hardware
5. Implemented as hardware framework (Reliability and Security Engine - RSE)
6. Hardware checks invoked by CHECK instructions embedded in application code

What is a detector?
A detector (or assertion) is a check on the value of a program variable or memory location at a particular execution point
Aim is to detect corruption of data values used in programs to ensure crash-failure semantics
Detectors’ properties in current set-up
- Involving functions of the current and previous value of the variable or location (namely, a[i] and a[i-1])
- Automatically derived based on the observed values at runtime
- Probabilistic model to choose rule and exception

Hardware Derivation
- Target Processor Description
- Configuration Information
- Optimal List of Detectors (class + parameters)
- Application Code

Hardware Implementation
Example: detector of class Alternate
Main components of the hardware implementation
1) Shadow Register File – holds the state of the checked locations
2) Assertion Table - stores the assertions’ parameters
3) Data-path - checks assertions independently of processor, by offloading assertion computation

www.iti.uiuc.edu
University of Illinois at Urbana-Champaign