Methodologies for Test Program Generation Exploiting Simulation Feedback

Giovanni Squillero
Politecnico di Torino
Torino, Italy

GOAL

• To propose a methodology for test-program generation based on simulation feedback for test and validation of microprocessors
GOAL

• To propose a broadly-applicable methodology for stimuli generation based on simulation feedback for test and validation of generic systems

Acknowledgements

• Danilo Ravotto
• Ernesto Sanchez
• Matteo Sonza Reorda
• Alberto Tonda

• + many others
Outline

• Proposed methodology
• Case studies
• Conclusions
**Proposed Methodology**

Stimuli Generator → Stimuli → System

Feedback

**Simulation-Based Approach**

- Uncovers problems by detecting incorrect behaviors when stimuli are applied
- Pros:
  - ...
- Cons:
  - Only takes into account a limited range of behaviors
  - *Never achieves 100% confidence*
Feedback-Based Approach

- Simulation-based approach
- Exploits feedback from simulation
- Incremental improvement/refinement of the solution (trial-and-error)
- Trade-off between computational resources and confidence
- May exploit heuristics or problem-specific knowledge

Proposed Methodology

- Exploit an Evolutionary Algorithm (EA) to generate stimuli to maximize a given function
  - Adaptative
  - Able to find unexpected solutions
  - Better than random
Evolutionary Algorithm

- Meta-heuristic optimization algorithm based on the concept of population and exploiting some principles of natural evolution

Evolutionary
Bio-Inspired
Computational Intelligence

Evolutionary Algorithm

- Trial and error
  - Solutions are iteratively refined
  - Local search
Evolutionary Algorithm

- Succession of random and deterministic steps
  - A systematic way of throwing dices
  - Better than pure random
- Population
  - Multiple solutions considered in each step
  - More resistant than pure hill-climbing
  - Different solutions may interbreed

Proposed Methodology

- Evolutionary Algorithm
- Stimuli
- System
- Fitness
Stimuli

• Valid assembly language programs
  – Syntax peculiarities
  – Instruction asymmetries
  – Subroutines/Interrupt handlers
  – Microprocessor specific features
    • Register window on SPARC
    • Global Descriptor Table and protected mode in IA86

Stimuli

• External world?
• In order to check a device (e.g., a I/O block) external stimuli must be taken into account
  – Highly correlated
  – Different formalism (e.g., data, waveform)
• Different types of stimuli
μGP (MicroGP)

• CAD Group general-purpose evolver
  — 3 versions (only 2 released under GPL)
  — Project started in 2002
  — 11 developers + contractors, students, ...
• Current version
  — ≈ 300 file, > 40,000 lines in C++
  — Improved similarity concept
    (late breaking paper @ GECCO 2008)
μGP Macro

\[ \text{add } r1, r2 \]
\[ r1 \in \{ \text{eax, ebx, ecx, edx} \} \]
\[ r2 \in \{ \text{eax, ebx, ecx, edx} \} \]

\[ r1 = \text{eax} \]
\[ r2 = \text{ebx} \]

\[ \text{add eax, ebx} \]
\[ \text{next} \]

\[ \text{op} = \text{add} \]
\[ op \in \{ \text{add, sub} \} \]
\[ r1 \in \{ \text{eax, ebx, ecx, edx} \} \]
\[ r2 \in \{ \text{eax, ebx, ecx, edx} \} \]

\[ \text{next} \]

\[ \text{op} = \text{add} \]
\[ op \in \{ \text{add, sub} \} \]
\[ r1 \in \{ \text{eax, ebx, ecx, edx} \} \]
\[ r2 \in \{ \text{eax, ebx, ecx, edx} \} \]

\[ \text{add eax, ebx} \]
\[ \text{next} \]
\[ jz \ $br \]
\[ \$br \in \{ \text{inner forward reference} \} \]

\[ jz \ n2371 \]

\[ \$\$$ = 175 \]
\[ \$\$$ = 2371 \]

\[ \text{push } \$r1 \]
\[ \text{movzx } \$r1, \text{BYTE } \$r2 + \$e1 \]
\[ \text{movzx } edx, \text{BYTE PTR}[\$r1 + \$e1] \]
\[ \text{cmp } ecx, \$e1 \]
\[ \text{pop } ecx \]
\[ \text{jg } \$j1 \]
\[ \text{cmp } ecx, 0 \]
\[ \text{pop } ecx \]
\[ \text{jg } \$j2 \]

\[ \ldots \]
http://ugp3.sourceforge.net/

MicroGP++ (aka. ugp3, µGP³)
- Information
- Download
- Credits

• System
  - The microprocessor
  - Helper module
Microprocessor

- Strongly problem dependent
- Model via simulation/emulation
  - HDL (netlist to high-level)
  - HW accelerated (e.g., exploiting FPGA)
  - Architectural simulator
  - ISA simulator
- Real device

Helper Module

- Usually a collection of scripts
- Applies stimuli
- Analyzes behavior
- Assembles fitness
  - e.g., a file containing a list of real numbers
Feedback Examples

- From simulation
  - Code coverage metrics (e.g., instruction coverage)
  - HW specific metrics (e.g., toggle coverage)
  - High-level information (e.g., FSM coverage)
- From running the real microprocessor
  - Performance counters
  - Physical measures (e.g., temperature, time, power consumption)

Outline

- Proposed methodology
- Case studies
- Conclusions
Types of Problem

• **Maximization**
  – Test (maximize FC%)
  – Verification (maximize tested functionalities)
  – ...

• **Needle in a haystack**
  – Find a counter-example
  – Find a bug
  – ...

Types of Problem

• **Transform** needle-in-a-haystack problems into maximization problems
• **Smooth** fitness landscape
• **Intermediate** goals
• **Heuristics**
• **Problem-specific** knowledge
• **Favor** exploration
Outline

- Proposed methodology
- Case studies
- Conclusions

Design Verification

- Generate test-programs for pre-silicon verification
- Verify that a microprocessor conforms to its specification
- Devise a set of programs able to activate all functionalities and corner cases
- Simulate the design against the reference model
System

- DLX/pII
  - Cleaned and simplified MIPS intended primarily for teaching purposes
  - 32-bit load/store architecture
  - 5-stage pipeline
  - VHDL RTL description

Feedback

- Code coverage metrics
  - Statement coverage (SC)
  - Branch coverage (BC)
  - Condition coverage (CC)
  - Expression coverage (EC)
- HW specific metric
  - Toggle coverage (TC)
System

- VHDL RTL
  - 4,558 statements
  - 3,695 branches
  - 193 conditional statements (1,764 expressions)
  - 8,283 logic bits

Experimental Results

- SC
- BC
- CC
- EC
- TC

Legend:
- Functional
- Random
- μGP
Post-silicon Verification

- Generate functional test programs for post-silicon verification
- The generated test programs
  - could be added as new content to improve existing validation suites
  - can be used to perform regression testing on future processor models
- Activity performed in collaboration with the ETM Group, Intel (Phoenix)

System

- Intel Pentium 4
  - 42-55 millions of transistors
  - 2GHz clock
  - NetBurst architecture
**System**

- Performance counters
  - Introduced in 1993 in IA32 architecture
  - P4 Counters architecture:
    - 48 event detectors
    - 18 event counters
    - 18 counter configuration control registers
  - Instruction-tagging (for discriminating non-speculative performance events)

---

**Post-silicon Verification**

- Use **performance monitors** as a proxy for the creation of certain micro-architectural events to
  - stress specific features
  - excite subtle corner cases
Mispredicted Ratio

- Maximize/minimize the ratio of mispredicted branches over the total branches
  - only non-speculative (retired) instructions are considered.
  - controlling the branch prediction rate is challenging
  - could generate interesting code fragments for exciting corner-case events
  - may cover flaws that would be hardly detected by manually-written targeted tests

<table>
<thead>
<tr>
<th>Program</th>
<th>#INST</th>
<th>Sampling Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
</tr>
<tr>
<td>Random [max]</td>
<td>278</td>
<td>6.01</td>
</tr>
<tr>
<td>Random [min]</td>
<td>426</td>
<td>0.10</td>
</tr>
<tr>
<td>Random [avg]</td>
<td>353.87</td>
<td>1.63</td>
</tr>
<tr>
<td>Random [std]</td>
<td>91.91</td>
<td>2.11</td>
</tr>
<tr>
<td>μGP (maximizing)</td>
<td>442</td>
<td>49.34</td>
</tr>
<tr>
<td>μGP (minimizing)</td>
<td>266</td>
<td>0.02</td>
</tr>
</tbody>
</table>
Trace Cache Deliver-Mode Ratio

- Max/Min ratio of clock cycles in which the trace cache is delivering µops to the execution unit instead of decoding or building traces.
  - intrinsic feature of the architectural design
  - test programs not biased by any specific solution
  - likely cover multiple cases, while an architect would target specific features
  - hard metric

<table>
<thead>
<tr>
<th>Program</th>
<th>#INST</th>
<th>Sampling Type</th>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random [max]</td>
<td>6</td>
<td>Time</td>
<td>99.32</td>
<td>98.13</td>
</tr>
<tr>
<td>Random [min(^\text{T})]</td>
<td>5</td>
<td>Time</td>
<td>85.39</td>
<td></td>
</tr>
<tr>
<td>Random [min(^\text{E})]</td>
<td>53</td>
<td>Event</td>
<td></td>
<td>81.46</td>
</tr>
<tr>
<td>Random [avg]</td>
<td>32.13</td>
<td></td>
<td>91.68</td>
<td>88.58</td>
</tr>
<tr>
<td>Random [std]</td>
<td>21.52</td>
<td></td>
<td>4.38</td>
<td>4.75</td>
</tr>
<tr>
<td>(\mu)GP (maximizing(^\text{T}))</td>
<td>36</td>
<td>Time</td>
<td>99.49</td>
<td></td>
</tr>
<tr>
<td>(\mu)GP (maximizing(^\text{E}))</td>
<td>40</td>
<td>Event</td>
<td></td>
<td>93.94</td>
</tr>
<tr>
<td>(\mu)GP (minimizing(^\text{T}))</td>
<td>5</td>
<td>Event</td>
<td>48.13</td>
<td></td>
</tr>
<tr>
<td>(\mu)GP (minimizing(^\text{E}))</td>
<td>55</td>
<td>Event</td>
<td></td>
<td>23.33</td>
</tr>
</tbody>
</table>
Time

• Constraints
  – 1,711 lines (239 macros)
  – Backed by Intel
• Set-up
  – Backed by Intel
• Experiments
  – 12h/program

SBST Test

• Software-Based Self Test
• Generate a set of test programs for the post-production test
• PLASMA (MIPS I)
  – 3-stage pipelined processor
  – Processor design models
    • Architectural level
    • RTL
    • Gate level (37K gates, 1,466 flip-flops)
Cumulative Methodology

- In each step a test-set is generated
- Already available programs are the starting point for the current step
- Incremental generation
  - Designer functional testbenches are exploited by the automatic tool
  - \(\mu\)GP test set is eventually exploited by the test engineer
Feedback

- Values of relevant metrics
- Straightforward definition
- Fitness and feedback fit together neatly
Experimental Results

Obsolescence

- Find bugs in the new implementation of a microprocessor against the original one (i.e., needle-in-a-haystack problem)
- Coupled with a hand-made pseudo-exhaustive verification process
- Activity performed in collaboration with TIMA
Solution

- Stressing all possible functionalities of the new microprocessor while comparing the behavior with the reference model
- Fitness strikingly different from the real goal

System

- Motorola/Freescale 68HC11
  - CISC microcontroller
  - Based on 6800 (upward compatible)
  - Two 8 bit accumulators (A & B) that can be concatenated to provide a 16 bit double accumulator (D)
  - Two 16 bit index registers (X & Y)
Experimental Results

PdT
- DAA: decimal adjust A
- SUBD: subtract memory from D
- SWI: software interrupt
- TXS: transfer accumulator X to stack
- TYS: transfer accumulator Y to stack
- TSX: transfer stack to accumulator X
- TSY: transfer stack to accumulator Y
- WAI: wait for interrupt

TIMA
- BSR: branch to subroutine

≈ 12 hours to create 24 small programs
Outline

- Proposed methodology
- Case studies
- Conclusions

Conclusions

- Simulation-based & feedback-based
- Evolutionary algorithm
- Broadly applicable
- Human resources
  - Limited (set-up)
- Computational resources
  - Easily parallelizable (generation)
  - Trade-off quality vs. effort
Conclusions

• May exploit other methodologies
  – Useful starting point
  – Effective completion

• May be coupled with other methodologies
  – Rule-based instruction randomizers
  – Simulation-based approaches