Instrumentation in Software Dynamic Translators for Self-Managed Systems

Bruce R. Childers
childers@cs.pitt.edu

Naveen Kumar, Jonathan Misurda and Mary Lou Soffa

Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA USA

Dept. of Computer Science
University of Virginia
Charlottesville, VA USA

http://www.cs.pitt.edu/coco
Introduction

• Self managed systems
  • Gather information online during execution
  • Make decisions to adapt to run-time conditions

• Software dynamic translation
  • Continuous compilation
  • Code security

• Monitoring needs program instrumentation

• Goal: Instrumentation to support range of program interception, monitoring and control activities
Approach: FIST

- **FIST**: Framework for Instrumentation in Software dynamic Translators
  - Describe & implement many algorithms
  - Abstractions for instrumentation

**Flexible**: Configurable for new uses, software dynamic translators & computing platforms (OS’s and machine architectures)

**Scalable**: Different amounts of information, at different levels of granularity, at different times
Preliminary Instrumentation Framework

- High-level instrumentation mapped to low level
- Instrumentation done on instructions (binary)

Diagram:
- High-Level Instrumentation
  - Source constructs
- Mappings
  - Relate low level to program
- Instruction-Level Monitoring & Control
  - Event-Response Model
- Instruction-Level Instrumentation
  - Dynamic Instrumentation
Event-Response Model

- **Events** monitor for run-time conditions
- **Responses** react to generated events

**Program**

- **Events**
  - User-defined conditions
  - **Instrumentation probes**
    - Static: Where to attach in program
    - Dynamic: Conditions to monitor

- **Responses**
  - User-defined event handlers
  - Data memory for information
  - Can be dynamic: Remove probes, insert new probes, new events
Dynamic Instrumentation

- Implements the dynamic conditions for events
- Instrumentation code injected

Dynamic instrumentation probe
- **Inline-hit-always**: Inserted before execution
- **Hit-once**: Run-time & removed immediately
- **Hit-many**: Run-time & removed explicitly

![Diagram of dynamic instrumentation process]

**Fast breakpoint**
Intercept program
Replace original instruction by a jump to dynamic check
FIST Instances

• Strata-SPARC
  • Binary dynamic translator
  • Many applications: Security, binary translation, monitoring, fault detection, etc.
  • SPARC RISC architecture & Solaris 9

• Jikes RVM/x86
  • Integrated at binary level on JIT’ed code
  • x86 CISC architecture & Linux

• Challenges: Instruction sets, memory for instrumentation code & data values, interactions with translators
### FIST for Strata-SPARC

<table>
<thead>
<tr>
<th>Application Binary</th>
<th>Integration with Strata</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Translator</td>
<td>• Translate does a callback</td>
</tr>
<tr>
<td>Memory Management</td>
<td>• Target independent services</td>
</tr>
<tr>
<td>Translated CodeCaching</td>
<td></td>
</tr>
<tr>
<td>Translated CodeLinking</td>
<td></td>
</tr>
<tr>
<td>Overhead Reduction</td>
<td>Instruction set – fixed length</td>
</tr>
<tr>
<td>Binary to RTL Translation</td>
<td>• Hit-many: Execute replaced instruction in probe</td>
</tr>
<tr>
<td>Dynamic Control Flow Graph</td>
<td>• Hit-once: Copy instruction back to original location</td>
</tr>
<tr>
<td>RTL Value and Dataflow Analysis</td>
<td>• Branches – delay slots</td>
</tr>
<tr>
<td>FIST</td>
<td>Instrumentation code &amp; data</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Strata Toolkit for building dynamic translators**

- Probe code dynamically generated into Strata’s translation cache
- Data memory allocated separately
FIST for Jikes RVM/x86

- Just-in-Time compiler from bytecode to x86
- FIST gains control on a new method load after JIT

- Instruction set – variable length
  - Execute replaced instruction in original location
  - Hit-many: Uses two fast breakpoints

```
probe:j probe
```

```
j reinsert

Saved Instruction
instruction
```
FIST for Jikes RVM/x86

- Multithreading
  - Instrumentation state across threads
  - Automatically managed data memory pool
  - Change data pool on thread switch

- Garbage Collection
  - Where to allocate pool & instrumentation code
  - GC may not be able to track accesses
  - Allocate separate memory buffer from OS
# Preliminary Experimental Results

## Strata-SPARC Instrumentation Probe Overhead

<table>
<thead>
<tr>
<th></th>
<th>Hit-once</th>
<th>Hit-many</th>
<th>Inline-hit-always</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time (ns)</strong></td>
<td>660</td>
<td>640</td>
<td>510</td>
</tr>
<tr>
<td><strong>Num. Instrs.</strong></td>
<td>72</td>
<td>53</td>
<td>49</td>
</tr>
</tbody>
</table>

**SPEC2K**: Effective avg. hit-many cost 102 ns

## Jikes/x86 Instrumentation Probe Overhead

<table>
<thead>
<tr>
<th></th>
<th>Hit-once</th>
<th>Hit-many</th>
<th>Inline-hit-always</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time (ns)</strong></td>
<td>469</td>
<td>939</td>
<td>65</td>
</tr>
<tr>
<td><strong>Num. Instrs.</strong></td>
<td>25</td>
<td>25</td>
<td>21</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>11</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

**SPECjvm98**: Effective avg. hit-many cost 1849 ns
Summary and Future Work

- **Flexible & scalable** binary-level instrumentation
- Demonstrated two instances
  - Strata-SPARC software dynamic translator
  - Jikes/x86 Java Research Virtual machine
- More self managing applications
- Language for instrumentation specification
- Translator to automatically map specification into instrumentation probes

- [http://www.cs.pitt.edu/coco](http://www.cs.pitt.edu/coco)
- childers@cs.pitt.edu