JVM Independent Replay in Java
RV’04 – April 3, 2004, Barcelona, Spain

Viktor Schuppan, Marcel Baur, Armin Biere
Computer Systems Institute, ETH Zürich

http://www.inf.ethz.ch/~schuppan/
Introduction

multi-threaded Java program

How am I supposed to find that bug?

What kind of GUI is that?

Why write another browser?

J. User

P. Developer

test tool

static checker

dynamic checker

evaluation trace

custom browser

test tool

static checker

dynamic checker

evaluation trace

custom browser

P. Developer

© 2004 V. Schuppan – Computer Systems Institute, ETH Zürich.
Introduction

→ Tool users work in familiar debugging environment
→ Tool developers focus on trace generation

Approach: bytecode instrumentation
Contents

1. Introduction

2. Trace Description

3. Results

4. Conclusion
Replay – Content-based Approach

Directly restore results of shared memory reads

[e.g. Pan, Linton 1988]
Restore partial order of shared memory accesses

directly restore order [e.g. LeBlanc, Mellor-Crummey 1987]
Restore partial order of shared memory accesses

restore thread switches [e.g. Russinovich, Cogswell 1996]
Replay – Comparison of Approaches

<table>
<thead>
<tr>
<th></th>
<th>trace size</th>
<th>parallelism</th>
<th>capture</th>
<th>replay</th>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>direct order</td>
<td>0/+</td>
<td>+</td>
<td>0</td>
<td>0/+</td>
<td>+</td>
</tr>
<tr>
<td>thread switches</td>
<td>+/0</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>0</td>
</tr>
</tbody>
</table>

(c) 2004 V. Schuppan – Computer Systems Institute, ETH Zürich.
### Replay – Comparison of Approaches

<table>
<thead>
<tr>
<th></th>
<th>trace size</th>
<th>parallelism</th>
<th>capture</th>
<th>replay</th>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-based</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>direct order</td>
<td>0/+</td>
<td>+</td>
<td>0</td>
<td>0/+</td>
<td>+</td>
</tr>
<tr>
<td>thread switches</td>
<td>+/0</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>0</td>
</tr>
</tbody>
</table>

© 2004 V. Schuppan – Computer Systems Institute, ETH Zürich.
Example:

```c
for (i = 0; i < 3; i++) {
    if (i % 2 == 0) {shared++;} 
    else {shared*=2;}
}
```

unroll:

```c
i = 0;
if (i < 3) {
    if (i % 2 == 0) {shared++;} 
    i++;
    if (i < 3) {
        if (i % 2 == 0) 
            else 
                {shared*=2;}
    }
    i++;
    if (i < 3) {
        /* replayer action */
        shared++; }
    i++;
```
Specifying Points in an Execution – 1

**Software instruction counter** [Mellor-Crummey, LeBlanc 1989]
(thread id, instruction, #backjumps)

- capture: count backjumps
- replay: count backjumps

→ less work for capture

**Count specific instructions**
(thread id, instruction, #executions)

- capture: count each instruction
- replay: count specific instructions

→ less work for replay
→ like debugger breakpoint
Specifying Points in an Execution – 1

**Software instruction counter** [Mellor-Crummey, LeBlanc 1989]
(thread id, instruction, #backjumps)

- capture: count backjumps
- replay: count backjumps

→ less work for capture

**Count specific instructions**
(thread id, instruction, #executions)

- capture: count each instruction
- replay: count specific instructions

→ less work for replay
→ like debugger breakpoint
Specifying Points in an Execution – 2

**after**
- trace: handle easily
- instrument before successors
- more natural?

**before**
- trace: may need to guess for last instruction
- instrument before instruction

- typically no difference in VM
Specifying Points in an Execution – 2

**after**

→ trace: handle easily
→ instrument before successors
→ more natural?

**before**

→ trace: may need to guess for last instruction
→ instrument before instruction

→ typically no difference in VM
Example – Producer/Consumer

// Producer
Method void run()
 0 goto 3
 3 invokevirtual notFull()
 6 ifeq 3
 9 iconst_0
10 invokevirtual put(int)
13 goto 3

// Consumer
Method void run()
 0 goto 3
 3 invokevirtual notEmpty()
 6 ifeq 3
 9 invokevirtual get()
12 istore_1
13 goto 3

# (incomplete) schedule
# 1 (producer) running
before Producer 1 13 1
switch 2 # c1
before Consumer 1 9 1
switch 3 # c2
before Consumer 1 13 1
switch 2 # c1
# error executing get

© 2004 V. Schuppan – Computer Systems Institute, ETH Zürich.
## Syntax

**Events**

- **before**
  - true just before specified point in execution
- **in**
  - true when in `wait`, `sleep`, `join` at specified point

**Actions**

- **switch**
  - switch thread
- **notify**
  - notify thread
- **timeout**
  - time-out thread
- **die**
  - wait for termination and switch thread
- **terminate**
  - terminate replay
- **log**
  - log message

**Control flow**

- **loopbegin**
  - start loop
- **loopend**
  - end loop

**Execute finite or infinite loop in schedule**
Notes on Implementation

**Approach**

- criteria: portability, maintenance, features
- choices: modify VM, use standard interface, instrument code

**Places to instrument**

- given by schedule and
- thread state related events

**Mechanics**

- use `wait/notify` to block/unblock a thread
  - get proper handling of (recursive) locks for free
- track thread state separately
public void block() throws Exception {
    synchronized(lock) {
        while (blocked) {
            try {lock.wait();}
            catch (InterruptedException e) 
                {/* report error */} 
        }
        blocked = true; }
} // block

public void unblock() {
    synchronized(lock) {
        blocked = false;
        lock.notifyAll(); }
} // unblock
Results

Portable replay

- on Sun’s VMs 1.3/1.4, Jikes, Kaffe, Kissme
- debugging with jdb, Eclipse, JDebugTool, and JSwat
- Java thread model?
  → interrupt{ed thread consumes notify?}

Overhead

- slowdown (Sun VM 1.4) typically < 10 times
- +7 instructions at each instrumented location

Capture

- use JNuke to capture benchmark runs
- implement listener for JPF with ~250 loc as a matter of 1 – 2 days
Conclusions

Suggest to use debuggers to browse traces generated by checkers

Propose format to describe multi-threaded execution traces

Show feasibility of portable replay

Thanks.
Keep out!
Backup slides
Thread Model

(existing)

(start)

(r. interrupted)

(new Thread())

(wait, join, sleep)

(interrupted)

(run return)

(w. notified)

(waiting)

(w. int. throw)

(w. timed-out)

(joining)

(j. joined)

(j. int. flag)

(j. int. throw)

(j. int. flag)

(sleeping)

(s. int. throw)

(s. int. flag)

(s. timed-out)

(running)

(interrupt)

(time-out)

(notify)

(interrupt notify)

(interrupt)

(died)

(throw InterruptedException)

(reacquire lock), reschedule/set interrupted flag

(reacquire lock), reschedule/throw InterruptedException

(reacquire lock), reschedule
Performance – Overhead

![Graph showing performance overhead for different benchmarks and systems. The x-axis represents benchmarks: Barrier, Sync, LUFact, Crypt, SOR, and SMM. The y-axis represents overhead in percentage. Different systems are compared: Sun 1.4, Sun 1.3, Jikes, Kaffe, and Sable. Each system is represented by a different color.](image_url)