Empowering Software Debugging Through Architectural Support for Program Rollback

Radu Teodorescu and Josep Torrellas
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu
Motivation

• Production software is hard to debug
• Need lightweight, continuous monitoring system

We propose: hardware/software approach:
• Architectural support for program undo
• Monitoring and recovery from bugs in production systems
Processor with program undo support

- Rollback/re-play of large code sections
- Very low overhead
- Speculation control:
  - In software: spec control instructions
  - In hardware: dynamic sliding window

Software control

Hardware control

Safe Code
Begin Spec
Speculative code
End Spec
Safe Code
Safe
Speculative
Contributions

• We implemented an FPGA-based prototype of a processor with undo support
• We show that simple hardware can provide powerful debugging tools
• We discuss possible applications to software debugging
• Initial assessment using buggy programs
Debugging Production Code

Original code

num=1;
...
p=m[a[*x]]+&y;
...
um++;

Instrumented code

num=1;
enter_spec();
p=m[a[*x]]+&y;
...
if(pstate()==REEXEC)
{
  info_collect();
}
exit_spec(flag);
um++;
Implementation

• Save/restore processor state:
  • Register checkpointing and restoration
  • Data cache that buffers speculative data (commit or invalidate)
  • Instructions enable/disable speculation on-the-fly
• Limits: cache size, I/O

(checkpointed state)

- CPU
- Data Cache
- Memory
Other uses of program rollback support
Code versioning

- Binary keeps two versions:
  - conservative - safer
  - aggressively optimized - potentially buggy
- Execute aggressive code speculatively
- If test fails, fall back on conservative version
Sandboxing OS drivers

- Buggy drivers - main cause of OS crashes
- Kernel survival in the presence of faulty drivers
- Execute driver code speculatively
- If crash, re-initialize driver
Failure-oblivious computing

- Enables applications to execute beyond some errors [Rinard04]
- Invalid memory accesses are caught
  - write: ignore, continue execution
  - read: manufacture value, continue
- After invalid access – speculative execution for a certain duration
Evaluation
Hardware prototype

- LEON2 - SPARC V8 compliant processor
- In-order, single issue, 5-stage pipeline
- Windowed register file
- L1 instruction and data caches
- Synthesizable, open source VHDL code
- Fully functional, runs Linux embedded
System Deployment

Processor Image

I/O Terminal

Binaries

Control App.

PCI

JTAG

COM
Evaluation

• Applications with known bugs
• Manually instrument the code
• Detection window contains:
  • bug location
  • bug manifestation
• Determine if we can roll back the buggy code section
## Buggy applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Bug Description</th>
<th>Successful rollback</th>
<th>Dynamic Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncompress-4.2.4</td>
<td>Input file name longer than 1024 bytes corrupts stack</td>
<td>Yes</td>
<td>10653</td>
</tr>
<tr>
<td>polymorph-0.4.0</td>
<td>Input file name longer than 2048 bytes corrupts stack</td>
<td>No</td>
<td>103838</td>
</tr>
<tr>
<td>tar-1.13.25</td>
<td>Unexpected loop bounds causes heap object overflow</td>
<td>Yes</td>
<td>193</td>
</tr>
<tr>
<td>man-1.5h1</td>
<td>Wrong bounds checking causes static object corruption</td>
<td>Yes</td>
<td>54217</td>
</tr>
<tr>
<td>gzip-1.2.4</td>
<td>Input file name longer than 1024 bytes overflows a global variable</td>
<td>Yes</td>
<td>17535</td>
</tr>
</tbody>
</table>
Conclusions

• Simple hardware can provide powerful debugging support

• We built an FPGA-based prototype of a processor with program undo support

• We describe a few possible applications to software debugging
Thank you!

Discussions and demo