Light64: Lightweight hardware support for data race detection during systematic testing

Adrian Nistor, Darko Marinov, Josep Torrellas

University of Illinois, Urbana – Champaign
http://iacoma.cs.uiuc.edu
Outline

Motivation
Systematic Testing
Light64
Evaluation
Conclusion
Data Races

- Common concurrency bug
- Difficult to detect
- Cause unexpected crashes even in code that is well tested
- Example:

  \[ X == 0 \]

  Thread A
  \[ X += 1 \]

  Thread B
  \[ X += 1 \]

  Depending on the run: \( X = 2 \) or \( X = 1 \)
**Contribution:** Light64

**Light64:** new data race detection technique

<table>
<thead>
<tr>
<th></th>
<th>Software</th>
<th>Light64</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware requirement</td>
<td>none</td>
<td>64 bits</td>
<td>72-400 Kbits</td>
</tr>
<tr>
<td>Execution overhead</td>
<td>8 X</td>
<td>1 – 37%</td>
<td>0.5%</td>
</tr>
</tbody>
</table>

**NO false positives**

**Detects 96% of races**
Outline

Motivation
Systematic Testing
Light64
Evaluation
Conclusion
Systematic Testing

- To detect bugs, we need high test coverage
- Very important in parallel programs
  - One input, many thread interleavings
- Systematic testing
  - Systematically execute many thread interleavings
  - Example: CHESS (used by Microsoft testers)
- Systematic testers include data race detection
- Turned off by default
  - Due to high runtime overhead
- **Light64:** Overhead low enough to be always ON
How Systematic Testing Works

SEGMENT == sequence of dynamic instructions

Execute many different interleavings
Multiplex segments in a uniprocessor
Outline

Motivation
Systematic Testing
Light64
Evaluation
Conclusion
Example

Race on $X$: because accesses to $X$ are not ordered by synchronization
The Idea

Perform two executions flipping the unordered segments
The Idea

If segments A1 and B1

- have NO race → they are independent → NOTHING changes
- have a RACE → we also flipped the race → Access history changes

Thread A

A1

A2

Thread B

B1

Sync → No race possible

No sync → May have a race

UN – FLIPPED

FLIPPED

PRESERVE
The Idea

If segments A1 and B1
have NO race → they are independent → NOTHING changes
have a RACE → we also flipped the race → SOMETHING changes

Light64
Nistor, Marinov, Torrellas
Overview

- Use two different executions
- Same synchronization order
  - If change ➔ know for sure there is a race
  - No change ➔ highly probable no race

Nistor, Marinov, Torrellas
Phases in **Light64**

- Detect if races exist
  - Fast, over all thread interleavings executed by the tester
  - Issues
    - How to detect deviations (e.g. from 0 to 3) \(\rightarrow\) HW hash
    - How to flip the segments with low overhead \(\rightarrow\) SW

- Pin – point races
  - Slow, classic data race detection algorithm
  - Only if there are races
  - Only for the racy interleavings
  - Optimization: only for selected racy interleavings

Nistor, Marinov, Torrellas
Detecting Deviations

- Per thread: hash all the values read from memory on-the-fly
- Compare hashes of two executions with same sync order
  - Different hashes ➔ know for sure there is a race
  - Identical hashes ➔ high probability no race
Example: Detecting Deviations

Thread A

Thread B

UN–FLIPPED

HASH (READs)

HASH (READs)

FLIPPED

HASH (READs)

HASH (READs)

END of execution

? =?= ?

Light64

Nistor, Marinov, Torrellas
BONUS

REGISTER

virtualize
migrate
context switch
no cache spills

Head of ROB

CRC 64 hash logic

64 bit register
Accumulates values read from memory

Light64

Nistor, Marinov, Torrellas
Flip with Low Overhead

STATE TREE

THREAD INTERLEAVING

Thread A

Thread B

A1
B1
A2
A3
B2

Nistor, Marinov, Torrellas
Flip with Low Overhead

Light64

Nistor, Marinov, Torrellas
Flip with Low Overhead

Piggy–back on Systematic Testing primitives to reduce overhead
Some synchronization orders are executed multiple times
Outline

Motivation
Systematic Testing
Light64

Evaluation

Conclusion
Experimental Setup

- Developed systematic tester in the lines of CHESS
- Tested all SPLASH-2 applications
- Run with 2 and 4 threads
- Execution overhead
  - Compare to a systematic tester with no race detection: Plain
- Accuracy
  - Compare to a systematic tester running a SW precise race detector
- Propose two Light64 versions:
  - Active: Aggressive flipping for high coverage
  - Passive: Modest flipping for minimum overhead
Execution Overhead (4 threads)

- Tradeoff execution overhead vs detection accuracy
  - Active: 37% overhead, 96% races detected
  - Passive: 2% overhead, 89% races detected
  - SW only: 8 X overhead, 100% races detected
### Detection Accuracy (4 threads)

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th></th>
<th>Inserted Races</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Light64</td>
<td>Precise SW</td>
<td>Light64</td>
</tr>
<tr>
<td>Barnes</td>
<td>311</td>
<td>311</td>
<td>192</td>
</tr>
<tr>
<td>Cholesky</td>
<td>4</td>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>FFT</td>
<td>0</td>
<td>0</td>
<td>42688</td>
</tr>
<tr>
<td>FMM</td>
<td>0</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>LU</td>
<td>0</td>
<td>0</td>
<td>15286</td>
</tr>
<tr>
<td>Ocean</td>
<td>2</td>
<td>2</td>
<td>45</td>
</tr>
<tr>
<td>Radiosity</td>
<td>12</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>Radix</td>
<td>0</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>Raytrace</td>
<td>7</td>
<td>7</td>
<td>87</td>
</tr>
<tr>
<td>Volrend</td>
<td>44</td>
<td>44</td>
<td>30</td>
</tr>
<tr>
<td>Water-NS</td>
<td>0</td>
<td>0</td>
<td>69</td>
</tr>
<tr>
<td>Water-SP</td>
<td>0</td>
<td>0</td>
<td>162</td>
</tr>
</tbody>
</table>

**Average race detection accuracy:** 96%
Also in the Paper

- Additional Light64 versions
- Optimization for the phase that pin-points races
- Characterization of the systematic testing
- Overhead and accuracy for two-threaded runs
- Additional results
- Software-only implementation
Outline

Motivation
Systematic Testing
Light64
Evaluation
Conclusion
## Conclusion

Introduced Light64, new data race detection technique for use during systematic testing.

<table>
<thead>
<tr>
<th></th>
<th>Light64</th>
<th>Other HW</th>
</tr>
</thead>
<tbody>
<tr>
<td>HW</td>
<td>64 bits</td>
<td>72 â€“ 400 Kbits</td>
</tr>
<tr>
<td>Virtualize</td>
<td>✓</td>
<td>NO or replicate the HW</td>
</tr>
<tr>
<td>Cache spill</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Context switch</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Migration</td>
<td>✓</td>
<td>NO or additional HW</td>
</tr>
<tr>
<td>No False Positives</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Few False Negatives</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Runtime Overhead</td>
<td>2 â€“ 37%</td>
<td>0.5%</td>
</tr>
</tbody>
</table>

Nistor, Marinov, Torrellas
Light64: Lightweight hardware support for data race detection during systematic testing

Adrian Nistor, Darko Marinov, Josep Torrellas

University of Illinois, Urbana – Champaign
http://iacoma.cs.uiuc.edu