Dynamically Detecting and Tolerating IF-Condition Data Races

Shanxiang Qi (Google),
Abdullah Muzahid (University of San Antonio),
Wonsun Ahn, Josep Torrellas
University of Illinois at Urbana-Champaign

HPCA-2014, Feb 2014
Background: Data Races

• A data race is a pair of concurrent (unordered) accesses where at least one is a write
• It is often a symptom of a concurrency bug
• Conventional data race detection
  – Happens-before: detect unordered accesses using a vector clock
  – Lock-set: detect concurrent accesses by comparing set of locks acquired by each thread
• Suffers from inaccuracy and high overhead
Motivating Example: Valgrind on FMM with 8 threads

- Inaccuracy discourages use by programmers
- High overhead lengthens debug cycle and precludes on-site deployment
Data Races in the Wild

- Studied characteristics of data races that were actually reported as concurrency bugs.
Data Races in the Wild

• Collected 54 races from open source bug libraries and reports
  ➡ servers
  ➡ desktop apps
  ➡ runtimes & libraries

<table>
<thead>
<tr>
<th>Apps.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>Web Server</td>
</tr>
<tr>
<td>MySQL</td>
<td>Database server</td>
</tr>
<tr>
<td>Mozilla</td>
<td>Browser</td>
</tr>
<tr>
<td>Pbzip2</td>
<td>Parallel bzip2</td>
</tr>
<tr>
<td>Redhat</td>
<td>glibc library</td>
</tr>
<tr>
<td>JAVA</td>
<td>SDK</td>
</tr>
</tbody>
</table>

38 out of 54 races were *IF-condition Data Races*
IF-Condition Data Race (ICR)

1. Modification of IF condition variables in the middle of IF body
2. Due to a racy write to the variable by another thread

if (p == q) {
    p = r;
    *p = x;
}

- Almost always a bug since it violates invariance of condition while executing control dependent code
  ➔ Almost no false positive bugs
- Very easy to pattern-match in the source code
  ➔ No need for profiling to insert runtime checks
- Amenable to low overhead detection
Contributions

• Identified a novel class of inherently harmful data races called IF-Condition Data Race (ICR)

• Proposed two new techniques for handling ICRs accurately and efficiently
  – SW-IF: Software-only implementation, ICR detection
  – HW-IF: Software + hardware implementation, ICR avoidance
SW-IF

• Main Idea:
  – Compiler inserts runtime checks to detect ICRs

• Two steps: Add Confirmation & Add Delay
  – Confirmation: Recomputation of IF condition at the end of the THEN and ELSE clauses to detect modification
  – Delay: (Optional) sleep to change timing during stress testing
SW-IF Example

T1  T2

(Optional) Delay  if (p == q) {
               usleep(15);
               p = r;
               *p = x;
               if (p != q)
                  printf(“bug!”);
            }

• Use:
  – Bug detection during the debug phase
  – Efficient enough to be used in production code
Adding Confirmations

- E – control expression
- E(L) – the set of all locations accessed in E
- E(SL) – the set of shared locations accessed in E
- In the example, E is (p == q), E(L) is {p, q}, and E(SL) is {p}

Instrumentation Rules:
- E(SL) should not be empty
- E should not contain write operations (since recomputation of E will cause side effects)
- Insert confirmations in the THEN and ELSE clauses: 1) at the end, or 2) before first write to E(L)

```
T1

if (p == q) {
    q = ...;
    *p = x;
    if (p != q)
        printf ("bug!");
}

T2

p = r;
```
HW-IF

• Main Idea:
  – Compiler marks shared locations in IF conditions for monitoring
  – HW prevents external accesses to monitored locations

• Add \textit{Watch} & \textit{Unwatch} for each location in E(SL)
  – \textit{Watch} instruction: Begins HW monitoring of location at start of IF body
  – \textit{Unwatch} instruction: Finishes HW monitoring of location at end of IF body
• Use:
  – Bug avoidance in production code

HW-IF Example

T1

Watch (p);
if (p == q) {
  *p = x;
}
Unwatch (p);

T2

p = r;
HW-IF Hardware Operation

Watch (\textit{var}); if (\textit{var})

\textbf{Tag (cache line addr)} \quad \textbf{Proc. ID}

| Cache line addr of \textit{var} | P1 |

\textbf{Address Watch Table (AWT)}
## Limitations of SW-IF and HW-IF

<table>
<thead>
<tr>
<th></th>
<th>False Negative (Failure to Detect ICR)</th>
<th>False Positive (Incorrect Detection of ICR)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SW-IF</strong></td>
<td>Occasional:</td>
<td>Very Rare (refer to paper)</td>
</tr>
<tr>
<td></td>
<td>• Writes in E prevent a confirmation from being inserted</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Writes to E(L) inside the THEN / ELSE clauses force confirmation to be placed early</td>
<td></td>
</tr>
<tr>
<td><strong>HW-IF</strong></td>
<td>Very Rare (refer to paper)</td>
<td>Harmless (since spurious Nacks only cause delays):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• False sharing in the AWT Nacks unrelated requests</td>
</tr>
</tbody>
</table>
Potential Bug Detection Capability

- Analyzed ICR bugs in our bug database of open source apps
- Estimate:
  - HW-IF detects 100% of bugs
  - SW-IF detects 47% of bugs
  ➔ Due to false negatives
Evaluation Setup

• Cetus source-to-source compiler
  – Instruments Confirmation & Delay, Watch & Unwatch

• SW-IF: Ran natively on Xeon multi-socket machine

• HW-IF: Ran on SESC simulator
  – Added 100-entry AWT
  – 8 processor CMP with snoopy MESI protocol

• Applications
  – For performance: SPLASH-2 with 4-8 threads
  – For bug detection capability: Cherokee and Pbzip
New ICR Bugs Detected

• Ran Cherokee and Pbzip with SW-IF and HW-IF

• HW-IF found 5 unreported bugs

• SW-IF found 3 of them
  – False negatives due to writes in IF condition
Execution Time Overhead of SW-IF

- Negligible average overhead: SW-IF (2%), SW-IFdelay (6%)
Execution Time Overhead of HW-IF

- HW-IF can avoid ICRs with negligible overhead of <1% on avg.
- Slight increase in overhead with more processors
Also in the paper

- Deadlock Handling
- Support for Context Switching
- Support for Multithreaded Processors
- Characterization of IF Statements in Applications
- Discussion on Double Checked Locking Bugs
Conclusion

• Identified a novel class of data races called IF-condition data races (ICRs)
  – Inherently harmful
  – Relatively frequent
  – Easy to pattern-match in the source code
  – Amenable to low overhead detection / avoidance

• Proposed two solutions that can be used for both development and production code
  – SW-IF: software-only solution to detect ICRs
  – HW-IF: software + hardware solution to avoid ICRs
Dynamically Detecting and Tolerating IF-Condition Data Races

Shanxiang Qi (Google), Abdullah Muzahid (University of San Antonio), **Wonsun Ahn**, Josep Torrellas
University of Illinois at Urbana-Champaign