Vulcan: Hardware Support for Detecting Sequential Consistency Violations Dynamically

Abdullah Muzahid, Shanxiang Qi, and Josep Torrellas
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

MICRO
December 2012
Sequential Consistency (SC)

- In SC, memory accesses:
  - Appear atomic
  - Have a total global order
  - For each thread, follow program order

<table>
<thead>
<tr>
<th>PA</th>
<th>PB</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0: x = 1</td>
<td>A0</td>
</tr>
<tr>
<td>A1: y = 1</td>
<td>A1</td>
</tr>
<tr>
<td>B0: p = y</td>
<td>B0</td>
</tr>
<tr>
<td>B1: t = x</td>
<td>B1</td>
</tr>
</tbody>
</table>
Sequential Consistency Violation (SCV)

- SCV: access reorder that does not conform to SC
- Machines support relaxed models, not SC
- Machines may induce SC violations (SCV)

Initially, $x=y=0$

In SC, if $p=1$ then $t=1$

**PA**

- A0: $x = 1$
- A1: $y = 1$

**PB**

- B0: $p = y$
- B1: $t = x$

```
A1
B0
B1
A0  p is 1
    t is 0
```

Very unintuitive bug
Example of SCV

T1
buf = malloc(...)  
init = true

T2
if (init)
... = buf[...]

Crash!!
When Can an SCV Occur?

- Two or more data races overlap
- They create a cycle

```
PA
A0: ref(x)  B0: ref(y)
A1: ref(y)  B1: ref(x)

fence

PB
A0: x =1   A1: y =1
B0: p =y   B1: t = x

fence
```
Why Detecting SCVs is Important?

- Programmers assume SC
  - SCV is almost **always a bug**: unexpected interleaving
  - Single-stepping debuggers **cannot reproduce** the bug
- Causes **portability** problems (e.g. Intel TBB)
  - Code may not work across machines
- Lock-free data structures sometimes explicitly use races but rely on SC
  - Traditional data race detectors won’t work
- Around **18%** of reported races can cause SCV (see paper)
Proposal: Vulcan

• Detects SCVs in relaxed consistency machines in highly precise manner
  • No false positives; no false negatives
• Provides info to debugger to debug SCV
• No SW changes; only use executable; negligible execution overhead
• Idea: Use HW to detect cycles of inter-thread dependences at runtime
• Approach:
  • Use the cache coherence protocol to dynamically record dependences
  • Interrupt the processor when a cycle is about to occur
Basic Algorithm

Region R2: Should not be the destination of a dependence from Region R1

Region R1: Should not be the source of a dependence to Region R2

Allowed Destination: AD > A0

Allowed Source: AS < B0

AS < min(B0, B1)

AS < B1

AD > A1

AD > max(A0, A1)
Hardware Structures

SN: Sequence Number
AS: Allowed Source
AD: Allowed Destination

N: # of processors

PA A0 B0 A1 B1 PB

0
1
0
1

N -1
Hardware Checks

3. Action at consumer
   If (\(S_{NBj} \geq AS_{Ai}[P_B]\))
   exception
   Else
   \(AD[P_B]\) of \(Ai\) and later = \(\max[\text{curr}_{\text{value}}, S_{NBj}]\)

2. Action at producer
   If (\(S_{NAi} \leq AD_{Bj}[P_A]\))
   exception
   Else
   \(AS[P_A]\) of \(Bj\) and earlier = \(\min[\text{curr}_{\text{value}}, S_{NAi}]\)
   All cases: Send response + \(S_{NBj}\)
Safe Accesses

• An access is **Safe** when it cannot cause an SCV anymore
  • The access and all its predecessors are **performed** and
  • All of disallowed destinations (in all the other procs) are **performed**
SC Violations and Safe Accesses

- When an SCV occurs the following must be true:
  - In the two arrows that form the cycle, the source reference is **Unsafe wrt the destination processor**
How Long to Keep Metadata?

- Keep metadata as long as the access can participate in an SCV
  - Keep metadata for Unsafe accesses only

Unsafe accesses = Pending + Disallowed_destinations_not_perf

- Store metadata in a per-processor SC Violation Queue (SCVQ)
  - Contains address + SN + AD[] + AS[], not data
Detecting Dependences and Cycles: Single Word Cache Line

- Inter-thread dependence induces a coherence bus transaction
- Bus transaction searches the SCVQs of other processors
- If hit, src and dst references exchange SN and run the Vulcan algorithm

Same for WAW, WAR
SC Violation Queue (SCVQ)

- Keeps Vulcan metadata for **Unsafe** local load/stores

- Need efficient search; **cannot rely on cache snooper**
- Counting bloom filter to **minimize useless SCVQ lookups**
Detecting Dependences and Cycles: Multiword Cache Line

- When the destination reference of an inter-thread dependence occurs…
  - Either coherence protocol triggers a coherence bus transaction
  - Or Vulcan forces a metadata update bus access

- Implementation: Vulcan adds V-State per byte in each line
  - Tracks whether the latest dependence on that word has already been recorded
  - If not recorded when processor accesses the word, even if no coherence action, force a metadata bus access

See Paper for Details
Issues

With these constraints

- No false positives, no false negatives

• Advantages:
  - Detects actual SC violations, not data races
  - Works for any memory model
  - Low overhead

• Limitations:
  - Race cycles involving only two processors (very large majority)
  - Not concerned with impact of compiler transformations on SCV
Modeled a multicore chip with 8 processors
- Core: Out of order, 2-issue width
- RC memory model
- Private L1, Shared L2
- Cache line size: 32 bytes
- Byte-level V-State bits
- SCVQ size: 256 entries

<table>
<thead>
<tr>
<th>PROGRAM</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dekker</td>
<td>Mutual exclusion</td>
</tr>
<tr>
<td>Lazylist</td>
<td>List-based concurrent set</td>
</tr>
<tr>
<td>Snark</td>
<td>Non-blocking double-ended queue</td>
</tr>
<tr>
<td>Harris</td>
<td>Non-blocking set</td>
</tr>
<tr>
<td>Pthread from glibc</td>
<td>Unwind code after canceling a thread</td>
</tr>
<tr>
<td>Crypt from glibc</td>
<td>Small table initialization code</td>
</tr>
<tr>
<td>DCL bug</td>
<td>Kernel using double-checked locking</td>
</tr>
<tr>
<td>SPLASH-2</td>
<td>8 programs from SPLASH-2</td>
</tr>
</tbody>
</table>
Vulcan Effectively Detects SCVs

<table>
<thead>
<tr>
<th>Program</th>
<th>SC Violations Found</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unique</td>
<td>Total</td>
<td>New?</td>
</tr>
<tr>
<td>Dekker</td>
<td>1</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>Lazylist</td>
<td>1</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>Snark</td>
<td>1</td>
<td>1467</td>
<td></td>
</tr>
<tr>
<td>Harris</td>
<td>1</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Pthread</td>
<td>2</td>
<td>142</td>
<td>Y</td>
</tr>
<tr>
<td>Crypt</td>
<td>2</td>
<td>130</td>
<td>Y</td>
</tr>
<tr>
<td>DCL</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>fmm (SPLASH2)</td>
<td>3</td>
<td>18</td>
<td>Y</td>
</tr>
</tbody>
</table>

- Vulcan detects 3 new bugs in important codes (libraries)
Example New Bug: Crypt Library Bug

- Found a new SCV in a bug fix

\[
\begin{align*}
\text{T1} & \quad \text{T2} \\
\text{if (init == False)} & \quad \text{if (init == True)} \\
\text{lock L} & \quad = \text{tab[...]} \\
\text{if (init == False)} & \quad = \text{init = True} \\
\text{tab[...]} = \ldots & \quad \text{fence} \\
\text{unlock L} & \quad \text{=} \text{tab[...]} \\
\end{align*}
\]

- Branch condition predicted TRUE although not TRUE
- THEN code uses the old tab[] (wrong one)
- Tab[] is updated
- Branch prediction is later confirmed correct

Crash!!!
Overhead

Traffic added:
- 9% due to piggybacked
- 12% due to extra bus accesses

Low overhead: OK for on-the-fly
Also in the Paper

• Full description of the protocol for multi-word cache lines
• Information that a debugger would get after the exception
• HW structure sizes and cost
• Comparison to related work
Conclusions

• SCV bugs are arguably the hardest type of concurrency bugs
• Vulcan is the first HW scheme to detect these bugs with high precision
  • No false positives; no false negatives
• It has low execution overhead for on-the-fly deployment
  • 6% for 8-proc runs; 4.4% for 4-proc runs
• It detects 3 previously unknown bugs in popular libraries

Lots of work to do!
Vulcan: Hardware Support for Detecting Sequential Consistency Violations Dynamically

Abdullah Muzahid, Shanxiang Qi, and Josep Torrellas
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

MICRO
December 2012