RADISH: Sound and Complete Race Detection in Software and Hardware

Joseph Devietti*
Benjamin P. Wood*
Karin Strauss**^
Luis Ceze*
Dan Grossman*
Shaz Qadeer^

*University of Washington
^Microsoft Research
Uses of Race Detection

- multithreaded record+replay
- simplifying consistency models
- atomicity checking
- atomicity enforcement
- testing & verification
- determinism checking
- determinism enforcement
- concurrency bug detection

many uses require sound+complete detection
software
Flanagan and Freund, PLDI 2009

hardware
Min and Choi, ASPLOS 1991
Muzahid et al., ISCA 2009
Prvulovic, HPCA 2006

complementary strengths

software:
- sound + complete static analysis
- slow polling-based

hardware:
- $ coherence event-based
- unsound / incomplete evictions descheduled threads
- $-line granularity
RADISH overview
sound+complete race detection in sw+hw

use sw to virtualize hw resources via “revision control”

byte-granularity tracking

hw mechanisms to reduce expensive broadcasts, sw lookups

unsound/incomplete
$ evictions
descheduled threads
$_line granularity
outline

happens-before

data race detection

full RADISH

results

conclusions
data races

Lamport, CACM 1978

2 concurrent accesses to the same memory location, \( \geq 1 \) of which is a write

unordered wrt the happens-before relation

transitive closure of program order + synchronization order
data races
Lamport, CACM 1978

At time $t_0$:
- $t_0$: Read X
- Release L

At time $t_1$:
- Read X
- Acquire L
- Write X

Program order:
- $po + sync$
- $data\ race$

Synchronization:
- $write\ X$
- $write\ X$

The diagram illustrates a data race due to concurrent operations on shared variables.
happens-before race detection

canonical sound + complete approach

Fidge, Computer 1991  Mattern, IWPDA 1989

per-thread metadata
read ordered with last write
write ordered with last write
and all last reads
per-location metadata

<table>
<thead>
<tr>
<th>thread</th>
<th>last synchronized with</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>t1@T, t2@U</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>location</th>
<th>last write</th>
<th>last reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>t2@T</td>
<td>t0@U, t1@-, t2@W</td>
</tr>
</tbody>
</table>
mapping to hardware

<table>
<thead>
<tr>
<th>thread</th>
<th>last synchronized with</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>t1 @U, t2 @T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>location</th>
<th>last write</th>
<th>last reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>t2@T</td>
<td>t0@U, t1@−, t2@W</td>
</tr>
</tbody>
</table>

unbounded # threads
unbounded # locations
Outline

Happens-before data race detection

Results

Conclusions
strawman

all metadata is in hw, so broadcast on every access?

local permissions cache what can be done without communication
local permissions

**READ, WRITE** or **NONE** permissions to each byte in a $ line

updated only on permissions violations and coherence events

$$\begin{array}{c}
ap0 \\
\text{write X} \\
ap1 \\
\text{write X} \\
ap2
\end{array}$$
strawman

metadata can be in hw or sw, so check sw on every access?

in-hardware status summarizes what metadata resides in hw
in-hardware status
what can we figure out without going to sw?

“checkout”
set IHS on checkout, degrade on $ evictions

evict write
Everything

evict read
All Last Reads

evict read
Last Write

evict write
In Software

“checkin”
also in the paper

leveraging type-safe languages to reduce metadata space overheads

asynchronous software lookups to reduce overheads
outline

happens-before
data race detection

in-$ \text{RADISH}$

full $\text{RADISH}$

results

conclusions
simulation methodology

- Pin-based simulator
- 8 cores, MESI coherence
- 8-way 64KB L1, 8-way 256KB private L2, 16-way 16MB L3
- PARSEC 2.1
- compare with FastTrack [Flanagan and Freund, PLDI 2009]
runtime compared to native

- blacksch
- fluid
- streamcl
- swaptions
- vips
- x264
runtime compared to FastTrack

- blacksch
- fluid
- streamcl
- swaptions
- vips
- x264
conclusions

sound+complete race detection in hw+sw

unmodified cache design

much faster than software-only race detection
thanks!