Demand-Driven Software Race Detection using Hardware Performance Counters

Joseph L. Greathouse†, Zhiqiang Ma‡, Matthew I. Frank‡
Ramesh Peri‡, Todd Austin†

†University of Michigan ‡Intel Corporation

CSCADS
Aug 2, 2011
Concurrency Bugs Still Matter

In spite of proposed hardware solutions

Hardware Data Race Recording

Bulk Memory Commits

Deterministic Execution/Replay

Sun Rock ?

AMD ASF ?
Concurrent Bugs Matter NOW

Thread 1
mylen=small
if(ptr==NULL)

Thread 2
mylen=large
if(ptr==NULL)

Nov. 2010 OpenSSL Security Flaw

len1=thread_local->mylen;
ptr=malloc(len1);
memcpy(ptr, data1, len1)

len2=thread_local->mylen;
ptr=malloc(len2);
memcpy(ptr, data2, len2)

ptr ∅ LEAKED
Speed up software race detection with existing hardware support.
Software Data Race Detection

- Add checks around every memory access
- Find inter-thread sharing events
- Synchronization between write-shared accesses?
  - No? Data race.
Example of Data Race Detection

Thread 1
mylen=small

if(ptr==NULL)
len1=thread_local(mylen);
ptr=malloc(len1);
memcpy(ptr, data1, len1)

Thread 2
mylen=large

if(ptr==NULL)
len2=thread_local(mylen);
ptr=malloc(len2);
memcpy(ptr, data2, len2)

Interleaved Synchronization?
SW Race Detection is Slow

Race Detector Slowdown (x)

Phoenix

PARSEC

Race Detector Slowdown (x)

GeoMean
Goal of this Work

Accelerate Software Data Race Detection

Technique #1: Making it Fast
Demand-Driven Data Race Detection

Technique #2: Keeping it Real
Find sharing events with existing HW
“Data races ... are failures in programs that access and update shared data in critical sections” – Netzer & Miller, 1992

```c
if(ptr==NULL)
    len1=thread_local->mylen;
    ptr=malloc(len1);
    memcpy(ptr, data1, len1);

if(ptr==NULL)
    len2=thread_local->mylen;
    ptr=malloc(len2);
    memcpy(ptr, data2, len2);
```
Very Little Inter-Thread Sharing

![Bar chart showing write-sharing events for Phoenix and PARSEC benchmarks.](chart.png)
Technique 1: Demand-Driven Analysis

Inter-thread Sharing Monitor

Software Race Detector

Inter-thread sharing
Inter-thread Sharing Monitor

- Check each memory op. for write-sharing
- Signal software race detector on sharing

- Possible to do in software
  - Can be built now with instrumentation
    - Slow. May take as long as race detection
Ideal Hardware Sharing Detector

- Follow read/write sets of threads

```
Thread 1
WRITE Y
```

```
Thread 2
READ Y
```

- Fast user-level faults

```
T1
R: ∅
W: {Y}
W->R

T2
R: ∅
W: ∅

Sharing Monitor
```

- Multi-threaded Application
- Software Race Detector
- Inter-thread Sharing Monitor
Limitations of Existing Hardware

- Fast faults
  - Solution: Enable detector for long periods of time

- Read/write sets
  - Solution:
Technique 2: Hardware Sharing Detector

- Hardware Performance Counters
  - Interrupt on cache coherency events
  - Intel’s HITM event: $W \rightarrow R$ Data Sharing

Limitations of this method:
- SMT sharing can’t be counted
- Cache eviction
- Others in paper
Demand-Driven Analysis on Real HW

Diagram:
- HITM Interrupt?:
  - NO → Execute Instruction → NO
  - YES → Enable Analysis
- Analysis Enabled?:
  - NO → Disable Analysis
  - YES → SW Race Detection
- Sharing Recently?:
  - NO → SW Race Detection
  - YES
Experimental Evaluation

- Modified Intel Inspector XE Race Detector
- Linux on 4-core Core i7, no Hyper-Threading

Performance Tests:
- Phoenix Suite
- PARSEC

Accuracy Tests:
- Phoenix Suite
- PARSEC
- Pre-release version of RADBench
Performance Difference

Race Detector Slowdown (x)

Phoenix

PARSEC

GeoMean
Performance Increases

Phoenix

PARSEC

Demand-driven Analysis Speedup (x)

histogram  kmeans  linear_regr.  matrix_mul.  pca  string_match  word_count  GeoMean  blackscholes  bodytrack  facesim  ferret  freqmine  raytrace  swaptions  fluidanimate  vips  x264  canneal  dedup  streamclus  GeoMean
Demand-Driven Analysis Accuracy

Accuracy vs. Continuous Analysis: 97%
Future Directions

- Better Performance
  - Fast user-level faults
  - Application specific hardware
- More Accuracy
  - Better performance counters
  - Inform SW on cache evictions/misses
- Smooth transition to ideal hardware

- Combine sampling & demand-driven analysis