On-Demand Dynamic Software Analysis

Joseph L. Greathouse
Ph.D. Candidate
Advanced Computer Architecture Laboratory
University of Michigan

December 12, 2011
Software Errors Abound

- NIST: SW errors cost U.S. ~$60 billion/year as of 2002
- FBI CCS: Security Issues $67 billion/year as of 2005
  - >\frac{1}{3} from viruses, network intrusion, etc.
Software Errors Abound

- NIST: SW errors cost U.S. ~$60 billion/year as of 2002
- FBI CCS: Security Issues $67 billion/year as of 2005
  - >\(\frac{1}{3}\) from viruses, network intrusion, etc.

**Cataloged Software Vulnerabilities**

![Graph showing the increase in cataloged software vulnerabilities from 2000 to 2008.](image-url)
Software Errors Abound

- NIST: SW errors cost U.S. ~$60 billion/year as of 2002
- FBI CCS: Security Issues $67 billion/year as of 2005
  - ⅓ from viruses, network intrusion, etc.

Adobe Warns of Critical Zero Day Vulnerability

Global Spam Drops by a Third After Rustock Botnet Gets Crushed, Symantec Says

Stuxnet attackers used 4 Windows zero-day exploits
Hardware Plays a Role
Hardware Plays a Role

In spite of proposed solutions

Hardware Data Race Recording

Bulk Memory Commits

Deterministic Execution/Replay

Bug-Free Memory Models

Atomicity Violation Detectors
In spite of proposed solutions

Hardware Plays a Role

- Hardware Data Race Recording
- Bulk Memory Commits
- Deterministic Execution/Replay
- SYSTEM MEMORY
- Bulk Memory
- Violation Detectors

TRANSACTIONAL MEMORY
In spite of proposed solutions

- Hardware Data Race Recording
- Bulk Memory Commits
- Deterministic Execution/Replay
- AMD ASF
- IBM BG/Q

Transactional Memory
Example of a Modern Bug

Nov. 2010 OpenSSL Security Flaw
Example of a Modern Bug

```c
if(ptr == NULL) {
    len=thread_local->mylen;
    ptr=malloc(len);
    memcpy(ptr, data, len);
}
```
Example of a Modern Bug

Thread 1
mylen=small

Thread 2
mylen=large

\[ \text{ptr} \rightarrow \emptyset \]
Example of a Modern Bug

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)

ptr
∅
Example of a Modern Bug

Thread 1
mylen=small

if(ptr==NULL)

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

(Thread 1)

Thread 2
mylen=large

if(ptr==NULL)

len2=thread_local->mylen;
ptr=malloc(len2);

mempcpy(ptr, data2, len2)

(ptr)

∅
Example of a Modern Bug

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)

ptr ∅
Example of a Modern Bug

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)

ptr
Example of a Modern Bug

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)

LEAKED
Example of a Modern Bug

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)

LEAKED

ptr

Thread 1
mylen=small

if(ptr==NULL)
Example of a Modern Bug

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)

ptr

LEAKED
Example of a Modern Bug

Thread 1  
mylen=small
  
if(ptr==NULL)
  
len1=thread_local->mylen;
ptr=malloc(len1);
mempy(ptr, data1, len1)

Thread 2  
mylen=large
  
if(ptr==NULL)
  
len2=thread_local->mylen;
ptr=malloc(len2);
mempy(ptr, data2, len2)

ptr

LEAKED
Dynamic Software Analysis

- Analyze the program as it runs
  - System state, find errors on any executed path
  - LARGE runtime overheads, only test one path

![Diagram of software analysis process with icons representing developer, program, analysis instrumentation, and test server(s).]
Dynamic Software Analysis

- Analyze the program as it runs
  + System state, find errors on any executed path
    - LARGE runtime overheads, only test one path
Dynamic Software Analysis

- Analyze the program as it runs
  + System state, find errors on any executed path
  - LARGE runtime overheads, only test one path

- Analysis Instrumentation
- Developer
- In-House Test Server(s)
- LONG run time
Dynamic Software Analysis

- Analyze the program as it runs
  + System state, find errors on any executed path
  - LARGE runtime overheads, only test one path
Runtime Overheads: How Large?

- Data Race Detection (e.g. Inspector XE) 2-300x
- Memory Checking (e.g. MemCheck) 5-50x
- Symbolic Execution 10-200x
- Taint Analysis (e.g. TaintCheck) 2-200x
- Dynamic Bounds Checking 10-80x
Could use Hardware

- Data Race Detection: HARD, CORD, etc.
- Taint Analysis: Raksha, FlexiTaint, etc.
- Bounds Checking: HardBound

- None Currently Exist; Bugs Are Here Now
- Single-Use Specialization
  - Won’t be built due to HW, power, verification costs
  - Unchangeable algorithms locked in HW
Goals of this Talk

- Accelerate SW Analyses Using Existing HW
- Run Tests **On Demand**: Only When Needed
- Explore Future **Generic HW Additions**
Outline

- Problem Statement

- Background Information
  - Demand-Driven Dynamic Dataflow Analysis

- Proposed Solutions
  - Demand-Driven Data Race Detection
  - Unlimited Hardware Watchpoints
Example Dynamic Dataflow Analysis

- Data
- Meta-data
- Input
Example Dynamic Dataflow Analysis

Data
Meta-data

Input

x = read_input()
Example Dynamic Dataflow Analysis

Data
Meta-data

Associate

\[ x = \text{read\_input()} \]
Example Dynamic Dataflow Analysis

\[ y = x \times 1024 \]

- \( x = \text{read\_input()} \)
- \( y = x \times 1024 \)
Example Dynamic Dataflow Analysis

```
a += y
z = y * 75
y = x * 1024
```

Input

```
x = read_input()
```

Output

```
a += y
z = y * 75
```
Example Dynamic Dataflow Analysis

```
a += y
z = y * 75
y = x * 1024
```

```
x = read_input()
```

```
validate(x)
```

```
a += y
z = y * 75
```

```
x = x * 1024
```

```
validate(x)
```

```
a += y
z = y * 75
```

```
validate(x)
```

```
Input
```

```
Data
```

```
Meta-data
```

```
Clear
```

```
Input
```

```
Data
```

```
Meta-data
```

```
x = read_input()
```

```
y = x * 1024
```

```
a += y
```

```
z = y * 75
```

```
validate(x)
```

```
Clear
```

```
Input
```

```
Data
```

```
Meta-data
```

```
x = x * 1024
```

```
validate(x)
```

```
Input
```

```
Data
```

```
Meta-data
```

```
x = read_input()
```

```
y = x * 1024
```

```
a += y
```

```
z = y * 75
```

```
validate(x)
```

```
Clear
```

```
Input
```

```
Data
```

```
Meta-data
```

```
x = read_input()
```

```
y = x * 1024
```

```
a += y
```

```
z = y * 75
```

```
validate(x)
```

```
Clear
```
Example Dynamic Dataflow Analysis

\[ a += y \]
\[ z = y \times 75 \]
\[ y = x \times 1024 \]
\[ w = x + 42 \]

Input

\[ x = \text{read\_input()} \]

\[ \text{validate}(x) \]

\[ y = x \times 1024 \]

\[ a += y \]

\[ z = y \times 75 \]

\[ w = x + 42 \]
Example Dynamic Dataflow Analysis

```
a += y
z = y * 75
y = x * 1024
w = x + 42
Check w
```

**Input**
- `x = read_input()`

**Data**
- `y = x * 1024`
- `z = y * 75`
- `a += y`

**Meta-data**
- `validate(x)`
- `w = x + 42`
Example Dynamic Dataflow Analysis

\begin{align*}
  a &\leftarrow y \\
  z &\leftarrow y \times 75 \\
  y &\leftarrow x \times 1024 \\
  w &\leftarrow x + 42
\end{align*}

Check \( w \)

Check \( a \)

Check \( z \)

Input

\( x = \text{read\_input}() \)

validate(x)
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Meta-Data Detection

Non-Shadowed Data
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

[Diagram showing the flow from Native Application to Instrumented Application with Meta-Data Detection]

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application → Instrumented Application

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Native Application

Instrumented Application

Meta-Data Detection
Finding Meta-Data

- No additional overhead when no meta-data
  - Needs hardware support
- Take a fault when touching shadowed data
- Solution: Virtual Memory Watchpoints
Finding Meta-Data

- No additional overhead when no meta-data
  - Needs hardware support
- Take a fault when touching shadowed data
- Solution: Virtual Memory Watchpoints
Results by Ho et al.

From “Practical Taint-Based Protection using Demand Emulation”

<table>
<thead>
<tr>
<th>System</th>
<th>Slowdown (normalized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taint Analysis</td>
<td>101.7x</td>
</tr>
<tr>
<td>On-Demand Taint Analysis</td>
<td>1.98x</td>
</tr>
</tbody>
</table>
Outline

- Problem Statement

- Background Information
  - Demand-Driven Dynamic Dataflow Analysis

- Proposed Solutions
  - Demand-Driven Data Race Detection
  - Unlimited Hardware Watchpoints
Software Data Race Detection

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

Shared?
Synchronized?
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)
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)
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)

Shared?
Example of Data Race Detection

Thread 1
mylen=small

if(ptr==NULL)  
len1=thread_local->mylen;
ptr=malloc(len1);
mempcpy(ptr, data1, len1);

Thread 2
mylen=large

if(ptr==NULL)  
len2=thread_local->mylen;
ptr=malloc(len2);
mempcpy(ptr, data2, len2);

Synchronized?
Example of Data Race Detection

Thread 1
mylen=small

Thread 2
mylen=large

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);

Synchronized?
SW Race Detection is Slow

Race Detector Slowdown (x)

<table>
<thead>
<tr>
<th></th>
<th>Phoenix</th>
<th>PARSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kmeans</td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear_regression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>matrix_multiply</td>
<td></td>
<td></td>
</tr>
<tr>
<td>string_match</td>
<td></td>
<td></td>
</tr>
<tr>
<td>word_count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GeoMean</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blackscholes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bodytrack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>facesim</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ferret</td>
<td></td>
<td></td>
</tr>
<tr>
<td>freqmine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>raytrace</td>
<td></td>
<td></td>
</tr>
<tr>
<td>swaptions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fluidanimate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vips</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x264</td>
<td></td>
<td></td>
</tr>
<tr>
<td>canneal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dedup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>streamcluster</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GeoMean</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Inter-thread Sharing is What’s Important

“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);
```
Inter-thread Sharing is What’s Important

“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);
```
Inter-thread Sharing is What’s Important

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

Thread-local data
NO SHARING

TIME
Inter-thread Sharing is What’s Important

“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);
```
Inter-thread Sharing is What’s Important

“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);
```
Inter-thread Sharing is What’s Important

“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

% Write-Sharing Events

<table>
<thead>
<tr>
<th></th>
<th>Phoenix</th>
<th>PARSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kmeans</td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear_regression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>matrix_multiply</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pca</td>
<td></td>
<td></td>
</tr>
<tr>
<td>string_match</td>
<td></td>
<td></td>
</tr>
<tr>
<td>word_count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blackscholes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bodytrack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>facesim</td>
<td></td>
<td></td>
</tr>
<tr>
<td>freqmine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>raytrace</td>
<td></td>
<td></td>
</tr>
<tr>
<td>swaptions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fluidanimate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vips</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x264</td>
<td></td>
<td></td>
</tr>
<tr>
<td>canneal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dedup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>streamcluster</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Use Demand-Driven Analysis!

Multi-threaded Application

Software Race Detector

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Multi-threaded Application

Software Race Detector

Local Access

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

- Multi-threaded Application
- Software Race Detector
- Local Access
- Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor

Software Race Detector

Inter-thread Sharing
Use Demand-Driven Analysis!

- Multi-threaded Application
  - Software Race Detector
  - Inter-thread sharing
  - Inter-thread Sharing Monitor

[Image of a video game scene and a switch labeled 'ON' and 'OFF']
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Multi-threaded Application

Local Access

Inter-thread Sharing Monitor

Software Race Detector
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor

Local Access

Software Race Detector

Multi-threaded Application

M
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

![Diagram showing virtual memory watchpoints with a fault indicated]

FAULT
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

Inter-Thread Sharing

FAULT
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

- ~100% of accesses cause page faults
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
  - ~100% of accesses cause page faults

- Granularity Gap
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
  - ~100% of accesses cause page faults

- Granularity Gap
- Per-process not per-thread
- Must go through the kernel on faults
- Syscalls for setting/removing meta-data
## Hardware Sharing Detector

- HITM in Cache: $W\rightarrow R$ Data Sharing

<table>
<thead>
<tr>
<th>Core 1</th>
<th>Core 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
</tr>
</tbody>
</table>
### Hardware Sharing Detector

- **HITM in Cache: W→R Data Sharing**

<table>
<thead>
<tr>
<th>Core 1</th>
<th>Core 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S</td>
</tr>
<tr>
<td><strong>Write Y=5</strong></td>
<td>I</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
</tr>
</tbody>
</table>

The image illustrates the sharing of data between two cores. Core 1 writes a value `Y=5` and then accesses `S`. Core 2 reads `S` and then accesses `I`. This demonstrates data sharing between the cores.
Hardware Sharing Detector

- HITM in Cache: $W \rightarrow R$ Data Sharing

![Diagram showing cache sharing between cores](image)
Hardware Sharing Detector

- HITM in Cache: $W \rightarrow R$ Data Sharing

![Diagram showing HITM in Cache]

- Hardware Performance Counters

![Diagram showing Hardware Performance Counters]

Perf. Ctrs:

- Pipeline: 0, 0, 0, 0
- Cache: 0, 0, 0, 0
Hardware Sharing Detector

- HITM in Cache: $W \rightarrow R$ Data Sharing

- Hardware Performance Counters

```
Core 1  | Core 2
---------|---------
S        | S
Y=5      | M
         | Read Y
         | HITM
```

```
Perf. Ctrs
---------
Pipeline: 1
Cache: 0
0
0

```
Hardware Sharing Detector

- HITM in Cache: $W\rightarrow R$ Data Sharing

- Hardware Performance Counters
Hardware Sharing Detector

- HITM in Cache: $W \rightarrow R$ Data Sharing

- Hardware Performance Counters
Hardware Sharing Detector

- HITM in Cache: W→R Data Sharing

- Hardware Performance Counters
Hardware Sharing Detector

- HITM in Cache: W→R Data Sharing

- Hardware Performance Counters
Hardware Sharing Detector

- HITM in Cache: \( W \rightarrow R \) Data Sharing

- Hardware Performance Counters

![Diagram showing core 1 and core 2 with a transaction involving cache sharing and performance counters.](image)
Potential Accuracy & Perf. Problems

- Limitations of Performance Counters
  - HITM only finds $W \rightarrow R$ Data Sharing
  - Hardware prefetcher events aren’t counted

- Limitations of Cache Events
  - SMT sharing can’t be counted
  - Cache eviction causes missed events
  - False sharing, etc…

- PEBS events still go through the kernel
On-Demand Analysis on Real HW

Execute Instruction

Analysis Enabled?

YES → SW Race Detection

Sharing Recently?

YES
On-Demand Analysis on Real HW

Execute Instruction

Analysis Enabled?

YES

SW Race Detection

NO

Sharing Recently?

YES

Disable Analysis
On-Demand Analysis on Real HW

Diagram:
- HITM Interrupt? (YES → Enable Analysis, NO → Analysis Enabled?)
- Analysis Enabled? (NO → Execute Instruction, YES → SW Race Detection)
- SW Race Detection (NO → Disable Analysis, YES → Sharing Recently?)
- Sharing Recently? (NO → Disable Analysis, YES → Analysis Enabled?)
- Analysis Enabled? (YES → Enable Analysis, NO → Execute Instruction)
On-Demand Analysis on Real HW

- Execute Instruction
- HITM Interrupt?
- Analysis Enabled?
- SW Race Detection
- Sharing Recently?
- Disable Analysis

Enable Analysis

YES

NO

YES

NO

YES

NO

YES
On-Demand Analysis on Real HW

1. HITM Interrupt?
   - NO
   - YES

2. Analysis Enabled?
   - NO
   - YES

3. Execute Instruction

4. SW Race Detection

5. Sharing Recently?
   - NO
   - YES

6. Enable Analysis

7. Disable Analysis

8. > 97%

9. < 3%
Performance Difference

Phoenix

PARSEC

Race Detector Slowdown (x)

histogram

kmeans

linear_regression

matrix_multiply

string_match

word_count

GeoMean

blackscholes

bodytrack

facesim

ferret

fremine

raytrace

swaptions

fluidanimate

evips

x264

canneal

dedup

streamcluster

GeoMean

99
Performance Increases

Demand-driven Analysis Speedup (x)

<table>
<thead>
<tr>
<th>Phoenix</th>
<th>PARSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>51x</td>
<td></td>
</tr>
</tbody>
</table>

- histogram
- kmeans
- linear_regression
- matrix_multiply
- string_match
- word_count
- blackscholes
- bodytrack
- facesim
- ferret
- freqmine
- raytrace
- raytrace
- fluidanimate
- vips
- x264
- canneal
- dedup
- streamcluster
- GeoMean
Demand-Driven Analysis Accuracy

![Bar chart showing demand-driven analysis speedup (x) for various benchmarks and GeoMean. The chart includes benchmarks such as histogram, kmeans, matrix_multiply, pca, matrix_multiply, GeoMean, blackscholes, bodytrack, facesim, ferret, raytrace, swaptions, fluidanimate, vips, x264, canmean, dedup, streamcluster, and GeoMean.]
Demand-Driven Analysis Accuracy

Accuracy vs. Continuous Analysis: 97%
Outline

- Problem Statement
- Background Information
  - Demand-Driven Dynamic Dataflow Analysis
- Proposed Solutions
  - Demand-Driven Data Race Detection
  - Unlimited Hardware Watchpoints
Watchpoints Globally Useful

- Byte/Word Accurate and Per-Thread
Watchpoints Globally Useful

- Byte/Word Accurate and Per-Thread
Watchpoints Globally Useful

- Byte/Word Accurate and Per-Thread
Watchpoints Globally Useful

- Byte/Word Accurate and Per-Thread
Watchpoints Globally Useful

- Byte/Word Accurate and Per-Thread
Watchpoints Globally Useful

- Byte/Word Accurate and Per-Thread
Watchpoint-Based Software Analyses

- Taint Analysis
- Data Race Detection
- Deterministic Execution
- Canary-Based Bounds Checking
- Speculative Program Optimization
- Hybrid Transactional Memory
Challenges

- Some analyses require watchpoint ranges
  - Better stored as base + length
- Some need large # of small watchpoints
  - Better stored as bitmaps
- Need a large number
The Best of Both Worlds

- Store Watchpoints in Main Memory

- Cache watchpoints on-chip
Demand-Driven Taint Analysis

\[ \text{Slowdown (x)} \]

- **MINEMU**
- **Umbra**
- **Watchpoint**

### Tools and Performance

- **164.zip**
- **175.vpr**
- **176.gcc**
- **181.mcf**
- **186.crafty**
- **197.parse**
- **252.eon**
- **253.perlbmk**
- **254.gap**
- **255.vortex**
- **256.bzip2**
- **300.twolf**
- **GeoMean**
Watchpoint-Based Data Race Detection

![Bar graphs comparing Phoenix and PARSEC benchmarks with speedup (x) on the y-axis and various benchmarks on the x-axis.](image-url)
Watchpoint Deterministic Execution

**Performance vs Grace**

- **Phoenix**
  - Performance vs Grace for various benchmarks:
    - histogram
    - kmeans
    - linear_regression
    - matrix_multiply
    - pca
    - string_match
    - word_count
    - GeoMean

- **SPEC OMP2001**
  - Performance vs Grace for various benchmarks:
    - 312.swim_m
    - 316.applu_m
    - 318.galtimore_m
    - 320.equake_m
    - 324.apsi_m
    - 326.gafort_m
    - 328.fma3d_m
    - 330.art_m
    - 332.ammp_m
    - GeoMean
BACKUP SLIDES