On-Demand Dynamic Software Analysis

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

November 29, 2011
Software Errors Abound

- NIST: SW errors cost U.S. ~$60 billion/year as of 2002
A problem has been detected and Windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x00000050 (0xFD3094C2, 0x00000001, 0xFBFE7617, 0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c
Software Errors Abound

- NIST: SW errors cost U.S. ~$60 billion/year as of 2002

```
A problem has been detected and Windows has been shut down to prevent damage
to your computer.
The problem seems to be caused by the following file: SPOMDCON.SYS
PAGE_FAULT_IN_NONPAGED_AREA
If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow
these steps:
Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.
If problems continue, disable or remove any newly installed hardware
or software. Disable DOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.
Technical Information:

## A
STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

### SPOMDCON.SYS - Address FBFE7617 base at FBD5000, DateStamp 3e6dd67c```
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.

A problem has been detected and Windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: SPOMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

If this is the first time you've seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

- Check to make sure any new hardware or software is properly installed.
- If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.
- If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing.
- If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical Information:

```
*** STOP: 0x00000050 (0xFD3094C2, 0x00000001, 0xFBE7627, 0x00000000)

*** SPOMDCON.SYS - Address FBE7627 base at FDFE5000, DateStamp 3e6dd67c
```
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

- CVE Candidates
- CERT Vulnerabilities


Values: 0, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000
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
Hardware Plays a Role

In spite of proposed solutions

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

Transactional Memory
In spite of proposed solutions

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

Hardware Plays a Role

- IBM BG/Q
- AMD ASF

In spite of proposed solutions
Example of Modern Bug

Nov. 2010 OpenSSL Security Flaw
Example of 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 \emptyset
Example of 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)
Example of Modern Bug

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

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

Thread 2
mylen=large

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

if(ptr==NULL)

memcpy(ptr, data2, len2)

ptr ∅
Example of Modern Bug

### Thread 1
mylen=small

```c
if(ptr==NULL)
    memcpy(ptr, data2, len2)
ptr = malloc(len2);
len1=thread_local->mylen;
memcpy(ptr, data1, len1)
```

### Thread 2
mylen=large

```c
if(ptr==NULL)
    len2=thread_local->mylen;
    ptr=malloc(len2);
```
Example of 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
Example of Modern Bug

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

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

memcpy(ptr, data2, len2)

LEAKED

ptr
Example of Modern Bug

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

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

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

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

memcpy(ptr, data2, len2)

TIME

ptr

LEAKED
Example of 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

TIME

ptr
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

![Diagram showing the process of dynamic software analysis with steps involving the developer, program, instrumented program, analysis instrumentation, and in-house 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

Developer

Analysis Instrumentation

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
Dynamic Dataflow Analysis

- **Associate** meta-data with program values
- **Propagate/Clear** meta-data while executing
- **Check** meta-data for safety & correctness
- **Forms** dataflows of meta/shadow information
Example: Taint Analysis

Input

Data

Meta-data
Example: Taint Analysis

\[ x = \text{read_input()} \]
Example: Taint Analysis

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

- Data
- Meta-data

Associate
Example: Taint Analysis

\[ y = x \times 1024 \]

Example: Taint Analysis

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

\[ y = x \times 1024 \]
Example: Taint Analysis

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

Input

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

\[
y = x \times 1024
\]

\[
a += y \\
z = y \times 75
\]
Example: Taint Analysis

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

\[
y = x \times 1024
\]

\[
a += y
\]

\[
z = y \times 75
\]
Example: Taint Analysis

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

Input

```
x = read_input()
```

validate(x)

```
y = x * 1024
```

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

```
w = x + 42
```
Example: Taint Analysis

Data
Meta-data

Input

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

\( y = x * 1024 \)

\( z = y * 75 \)

\( a += y \)

\( w = x + 42 \)

validate(\( x \))

Check \( w \)
Example: Taint Analysis

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

---

Input

```
x = read_input()
```

validate(x)

Check w

```
Check a
Check z
```
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

Non-Shadowed Data

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

![Diagram showing Native Application, Instrumented Application, and Meta-Data Detection with Shadowed Data flow]
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

![Diagram showing comparison between Native Application and Instrumented Application]

Meta-Data Detection
Demand-Driven Dataflow Analysis

- Only Analyze Shadowed Data

Metadatas 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

No meta-data

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
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
Finding Meta-Data

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

\[ V \rightarrow P \]
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
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.

- **Imbench Best Case Results:**

<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>

- **Results when everything is tainted:**

![Graph showing slowdown for different systems: netcat_transmit, netcat_receive, ssh_transmit, ssh_receive.](image-url)
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.
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)
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)
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)

ptr write-shared?

Example of Data Race Detection
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)

ptr write-shared?
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

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

Interleaved Synchronization?
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

<table>
<thead>
<tr>
<th></th>
<th>Phoenix</th>
<th>PARSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Race Detector Slowdown (x)

- histogram
- kmeans
- linear_regression
- matrix_multiply
- pca
- string_match
- GeoMean
- blackscholes
- bodytrack
- facesim
- ferret
- fregmine
- raytrace
- raytrace
- raytrace
- fluidanimate
- vips
- x264
- canneal
- dedup
- streamcluster
- GeoMean
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

```
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);
```
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);
}
```

```c
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)

TIME

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);

Thread-local data
NO SHARING

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

Shared data
NO INTER-THREAD SHARING EVENTS
```
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);

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

<table>
<thead>
<tr>
<th>% Write-Sharing Events</th>
<th>Phoenix</th>
<th>PARSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>kmeans</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>linear_regression</td>
<td>0</td>
<td>1.5</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>1.5</td>
<td>2.0</td>
</tr>
<tr>
<td>pca</td>
<td>0</td>
<td>2.5</td>
</tr>
<tr>
<td>string_match</td>
<td>0</td>
<td>3.0</td>
</tr>
<tr>
<td>word_count</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>blackscholes</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>bodytrack</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>facesim</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>freqmine</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>raytrace</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>swaptions</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>fluidanimate</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>vips</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>x264</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>canneal</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>dedup</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>streamcluster</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

75
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

Inter-thread Sharing Monitor

Software Race Detector
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor

Software Race Detector

Inter-thread sharing

Multi-threaded Application

Race Detector

Inter-thread sharing
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor
Use Demand-Driven Analysis!

Inter-thread Sharing Monitor

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

- Multi-threaded Application
- Local Access
- Inter-thread Sharing Monitor
- Software Race Detector

Local Access
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?

![Diagram](image.png)
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
Finding Inter-thread Sharing

- Virtual Memory Watchpoints?
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

- Hardware Performance Counters

<table>
<thead>
<tr>
<th></th>
<th>Pipeline</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perf. Ctrs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Hardware Sharing Detector

- Hardware Performance Counters

Pipeline

Cache

Perf. Ctrs

1

0

0

0
Hardware Sharing Detector

- Hardware Performance Counters

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Perf. Ctrs

- 2
- 0
- 0
- 0
Hardware Sharing Detector

- Hardware Performance Counters

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>Cache</th>
</tr>
</thead>
</table>

Perf. Ctrs

- 2
- 0
- 1
- 0
Hardware Sharing Detector

- Hardware Performance Counters

<table>
<thead>
<tr>
<th></th>
<th>Perf. Ctrs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>-1</td>
</tr>
<tr>
<td>Cache</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Hardware Sharing Detector

- Hardware Performance Counters

Pipeline

Cache

Perf. Ctrs

2

-1

1

0

PEBS

Debug Store

FAULT
## Hardware Sharing Detector

### Hardware Performance Counters

<table>
<thead>
<tr>
<th></th>
<th>Perf. Ctrs</th>
<th>PEBS</th>
<th>Debug Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>2</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
## Hardware Sharing Detector

### Hardware Performance Counters

<table>
<thead>
<tr>
<th>Pipeline</th>
<th>Cache</th>
<th>Perf. Ctrs</th>
<th>PEBS</th>
<th>Debug Store</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1</td>
<td>Armed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
## Hardware Sharing Detector

### Hardware Performance Counters

<table>
<thead>
<tr>
<th></th>
<th>Perf. Ctrs</th>
<th>PEBS</th>
<th>Debug Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>2</td>
<td>-</td>
<td>EFLAGS</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Armed</td>
<td>EIP</td>
</tr>
<tr>
<td>Cache</td>
<td>1</td>
<td>-</td>
<td>RegVals</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>-</td>
<td>MemInfo</td>
</tr>
</tbody>
</table>

**Precise Fault**
Hardware Sharing Detector

- Hardware Performance Counters

- Intel’s HITM event: \(W \rightarrow R\) Data Sharing
Hardware Sharing Detector

- **Hardware Performance Counters**

- **Intel’s HITM event: W→R Data Sharing**
Hardware Sharing Detector

- Hardware Performance Counters

- Intel’s HITM event: \( W \rightarrow R \) Data Sharing
Potential Accuracy & Perf. Problems

- Limitations of Performance Counters
  - HITM only finds W→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
Demand-Driven Analysis on Real HW

Execute Instruction
Demand-Driven Analysis on Real HW

Execute Instruction

Analysis Enabled?
Demand-Driven Analysis on Real HW

- Execute Instruction
- Analysis Enabled?
- YES
- SW Race Detection
Demand-Driven Analysis on Real HW

Execute Instruction

Analysis Enabled?

YES

SW Race Detection

Sharing Recently?
Demand-Driven Analysis on Real HW

Execute Instruction

Analysis Enabled?

YES

SW Race Detection

Sharing Recently?

YES

YES
Demand-Driven Analysis on Real HW

- Execute Instruction
- Analysis Enabled?
  - YES: SW Race Detection
  - NO: Disable Analysis
- Sharing Recently?
  - YES
- Recently?
  - NO
  - YES
Demand-Driven Analysis on Real HW

- Execute Instruction
- Analysis Enabled?
  - YES
  - NO
  - HITM Interrupt?
    - NO
    - SW Race Detection
      - NO
      - YES
      - Disable Analysis
    - YES
    - NO
    - Sharing Recently?
      - NO
      - YES
Demand-Driven Analysis on Real HW

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

- NO

- HITM Interrupt?
  - YES
  - NO

- NO
Demand-Driven Analysis on Real HW

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

NO: Enable Analysis
YES: NO

NO: NO
YES: YES
Demand-Driven Analysis on Real HW

- Execute Instruction
- Analysis Enabled?
- Disable Analysis
- Sharing Recently?
- Enable Analysis
- SW Race Detection
- HITM Interrupt?
- NO
- YES
- NO

114
Performance Increases

Demand-driven Analysis Speedup (x)

Phoenix

PARSEC

histogram  kmeans  matrix_multiply  string  word_count  GeoMean  blackscholes  bodytrack  facesim  ferret  freqmine  raytrace  swaptions  fluidanimate  vips  x264  canneal  dedup  streamcluster  GeoMean
Performance Increases

![Graph showing performance increases for Phoenix and PARSEC](image)

Speedup (x) vs Demand-driven Analysis

- Phoenix: 51x

Software Benchmarks:
- histogram
- kmeans
- linear_regression
- matrix_multiply
- pca
- string
- match
- word_count
- GeoMean
- blackscholes
- bodytrack
- facesim
- ferret
- freqmine
- raytrace
- swaptions
- fluidanimate
- vips
- x264
- canneal
dedup
- streamcluster
- GeoMean
Demand-Driven Analysis Accuracy

The chart shows the demand-driven analysis speedup for various benchmarks, with the GeoMean at the bottom right of the chart.
Demand-Driven Analysis Accuracy

![Graph showing speedup for various benchmarks](image)
Demand-Driven Analysis Accuracy

The graph shows a comparison of demand-driven analysis accuracy across various benchmarks. The x-axis represents the demand-driven analysis speedup (x), and the y-axis shows the demand-driven analysis. The benchmarks include:

- histogram
- kmeans
- linear_regression
- matrix_multiply
- pca
- string_match
- word_count
- GeoMean
- blackscholes
- bodytrack
- facesim
- ferret
- freqmine
- raytrace
- swaptions
- fluidanimate
- vips
- x264
- canneal
- dedup
- streamcluster
- GeoMean

The data points indicate the speedup and accuracy for each benchmark, with some highlighted for emphasis.
Demand-Driven Analysis Accuracy

Demand-driven Analysis Speedup (x)

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

1/1
2/4
3/3
4/4
3/3
4/4
4/4
2/4
4/4
4/4
### Demand-Driven Analysis Accuracy

#### Speedup (x)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GeoMean</td>
<td>8.7</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
</tbody>
</table>

#### Accuracy vs. Continuous Analysis: 97%

- histogram, kmeans, linear_regression, matrix_multiply, string_match, word_count, GeoMean
- blackscholes, bodytrack, facesim, ferret, fregmine, raytrace, swaptions, fluidanimate, vips, x264, canneal, dedup, streamcluster, GeoMean
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 / Dynamic dataflow 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
Challenges

- Some analyses require watchpoint ranges
  - Better stored as base + length

- Some need large # of small watchpoints
  - Better stored as bitmaps
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

![Graph showing slowdowns for various benchmarks with MINEMU, Umbra, and Watchpoint categories.](image)

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

**Benchmarks:**
- 164.gzip
- 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

Phoenix

PARSEC
Watchpoint-Based Grace System

-performance vs grace

-Phoenix

- SPEC OMP2001

- 312. swim_m
- 316. applu_m
- 320. eqquake_m
- 324. apsi_m
- 326. gafort_m
- 328. fma3d_m
- 330. art_m
- 332. ammp_m
- GeoMean
BACKUP SLIDES
Performance Difference

<table>
<thead>
<tr>
<th>Phoenix</th>
<th>PARSEC</th>
</tr>
</thead>
</table>

Race Detector Slowdown (x)

![Graph showing performance difference between Phoenix and PARSEC](image-url)
## Accuracy on Real Hardware

<table>
<thead>
<tr>
<th></th>
<th>kmeans</th>
<th>facesim</th>
<th>ferret</th>
<th>freqmine</th>
<th>vips</th>
<th>x264</th>
<th>streamcluster</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>W→W</strong></td>
<td>1/1 (100%)</td>
<td>0/1 (0%)</td>
<td>-</td>
<td>-</td>
<td>1/1 (100%)</td>
<td>-</td>
<td>1/1 (100%)</td>
</tr>
<tr>
<td><strong>R→W</strong></td>
<td>-</td>
<td>0/1 (0%)</td>
<td>2/2 (100%)</td>
<td>2/2 (100%)</td>
<td>1/1 (100%)</td>
<td>3/3 (100%)</td>
<td>1/1 (100%)</td>
</tr>
<tr>
<td><strong>W→R</strong></td>
<td>-</td>
<td>2/2 (100%)</td>
<td>1/1 (100%)</td>
<td>2/2 (100%)</td>
<td>1/1 (100%)</td>
<td>3/3 (100%)</td>
<td>1/1 (100%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Spider Monkey-0</th>
<th>Spider Monkey-1</th>
<th>Spider Monkey-2</th>
<th>NSPR-1</th>
<th>Memcached-1</th>
<th>Apache-1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>W→W</strong></td>
<td>9/9 (100%)</td>
<td>1/1 (100%)</td>
<td>1/1 (100%)</td>
<td>3/3 (100%)</td>
<td>-</td>
<td>1/1 (100%)</td>
</tr>
<tr>
<td><strong>R→W</strong></td>
<td>3/3 (100%)</td>
<td>-</td>
<td>1/1 (100%)</td>
<td>1/1 (100%)</td>
<td>1/1 (100%)</td>
<td>7/7 (100%)</td>
</tr>
<tr>
<td><strong>W→R</strong></td>
<td>8/8 (100%)</td>
<td>1/1 (100%)</td>
<td>2/2 (100%)</td>
<td>4/4 (100%)</td>
<td>-</td>
<td>2/2 (100%)</td>
</tr>
</tbody>
</table>