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
Concurrency Bugs Still Matter
Concurrency Bugs Still Matter

In spite of proposed hardware solutions

- Hardware Data Race Recording
- Bulk Memory Commits
- Deterministic Execution/Replay
- Bug-Free Memory Models
- Atomicity Violation Detectors
Concurrenchy Bugs Still Matter

In spite of proposed hardware solutions

Hardware Data Race Recording

Bulk Memory Commits

Deterministic Execution/Replay

Bulk Memory Violation Detectors

Transactional Memory
In spite of proposed hardware solutions

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

**TRANSACTIONAL MEMORY**

- Sun Rock ?
- AMD ASF ?

---

**Concurrency Bugs Still Matter**
Concurrency Bugs Matter NOW

Nov. 2010 OpenSSL Security Flaw
Concurrency Bugs Matter **NOW**

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
∅
Concurrency Bugs Matter **NOW**

**Thread 1**
mylen = small

```c
if(ptr == NULL)
    memcpy(ptr, data2, len2)
```

**Thread 2**
mylen = large

```c
if(ptr == NULL)
    len2 = thread_local->mylen;
    ptr = malloc(len2);

memcpy(ptr, data1, len1);
```

**TIME**

```
ptr
∅
```
Concurrency Bugs Matter **NOW**

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)

ptr ∅
Concurrency Bugs Matter **NOW**

**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)
Concurrency Bugs Matter **NOW**

**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)

**TIME**

ptr

LEAKED
Concurrency Bugs Matter NOW

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
Concurrency Bugs Matter **NOW**

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)

TIME

LEAKED
Concurrency Bugs Matter NOW

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
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)
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
Example of Data Race Detection

Thread 1
mylen=small

1. if(ptr==NULL)
2. len1=thread_local->mylen;
3. ptr=malloc(len1);
4. memcpy(ptr, data1, len1)

Thread 2
mylen=large

1. if(ptr==NULL)
2. len2=thread_local->mylen;
3. ptr=malloc(len2);
4. 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

ptr write-shared?

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)

Interleaved Synchronization?
Example of Data Race Detection

Thread 1
mylen=small

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

Thread 2
mylen=large

if(ptr==NULL)
len2=thread_local->mylen;
ptr=malloc(len2);
mempcpy(ptr, data2, len2)
SW Race Detection is Slow

![Graph showing race detector slowdown for Phoenix and PARSEC]
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
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

```
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)
```

Thread-local data

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

NO SHARING
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

```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)

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

```
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

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

<table>
<thead>
<tr>
<th></th>
<th>Phoenix</th>
<th>PARSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>kmeans</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>linear_regression</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pca</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>string_match</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>word_count</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>blackscholes</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bodytrack</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>facesim</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>freqmine</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>raytrace</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>swaptions</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>vips</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x264</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>canneal</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>dedup</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>streamcluster</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Very Little Inter-Thread Sharing

% Write-Sharing Events

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

0 0.5 1 1.5 2 2.5 3

Very Little Inter-Thread Sharing
Technique 1: Demand-Driven Analysis

- Multi-threaded Application
- Software Race Detector

Inter-thread Sharing Monitor
Technique 1: Demand-Driven Analysis

Multi-threaded Application

Software Race Detector

Local Access

Inter-thread Sharing Monitor
Technique 1: Demand-Driven Analysis

![Diagram showing multi-threaded application, local access, and inter-thread sharing monitor.]

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

**Diagram Elements:**
- Mapped to the context provided.
- Clear and detailed visualization.

**Key Points:**
- Demand-driven analysis is crucial for identifying inefficiencies in multi-threaded applications.
- Local access helps in maintaining data integrity and reduces potential race conditions.
- The inter-thread sharing monitor facilitates efficient synchronization and resource management.
- Software race detectors are essential for detecting and resolving concurrency issues in real-time.
Technique 1: Demand-Driven Analysis

Multi-threaded Application Inter-thread sharing Inter-thread Sharing Monitor

Software Race Detector
Technique 1: Demand-Driven Analysis

Software Race Detector

Inter-thread sharing

Inter-thread Sharing Monitor
Technique 1: Demand-Driven Analysis

Inter-thread Sharing Monitor

Software Race Detector
Technique 1: Demand-Driven Analysis

Multi-threaded Application

Software Race Detector

Inter-thread Sharing Monitor
Technique 1: Demand-Driven Analysis

Inter-thread Sharing Monitor

Software Race Detector

Local Access
Technique 1: Demand-Driven Analysis

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

Local Access
Technique 1: Demand-Driven Analysis

Multi-threaded Application

Software Race Detector

Inter-thread Sharing Monitor
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

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE Y</td>
<td>READ Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sharing Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>T1</strong></td>
</tr>
<tr>
<td>R: ∅</td>
</tr>
<tr>
<td>W: ∅</td>
</tr>
<tr>
<td><strong>T2</strong></td>
</tr>
<tr>
<td>R: ∅</td>
</tr>
<tr>
<td>W: ∅</td>
</tr>
</tbody>
</table>
Follow read/write sets of threads

Thread 1
WRITE Y

Thread 2
READ Y

Sharing Monitor

T1
R: ∅
W: {Y}

T2
R: ∅
W: ∅
Ideal Hardware Sharing Detector

- Follow read/write sets of threads

Thread 1
WRITE Y

Thread 2
READ Y

Sharing Monitor

T1
R: ∅  
W: {Y}

T2
R: ∅  
W: ∅

W->R  Sharing
Ideal Hardware Sharing Detector

- Follow read/write sets of threads

  Thread 1
  WRITE Y

  sharing monitor

  T1
  R: ∅
  W: {Y}

  T2
  R: ∅
  W: ∅

- Fast user-level faults

Multi-threaded Application

Software Race Detector

Inter-thread Sharing Monitor
Ideal Hardware Sharing Detector

- Follow read/write sets of threads

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>WRITE Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>W-&gt;R</td>
</tr>
<tr>
<td>R: ∅</td>
<td>W: {Y}</td>
</tr>
<tr>
<td>W: ∅</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thread 2</th>
<th>READ Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>R: ∅</td>
</tr>
<tr>
<td>W: ∅</td>
<td></td>
</tr>
</tbody>
</table>

- Fast user-level faults

Multi-threaded Application
Software Race Detector
Inter-thread Sharing Monitor
Ideal Hardware Sharing Detector

- Follow read/write sets of threads
  
<table>
<thead>
<tr>
<th>Sharing Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
</tr>
<tr>
<td>W: {Y}</td>
</tr>
<tr>
<td>R: ∅</td>
</tr>
<tr>
<td>W-&gt;R Sharing</td>
</tr>
<tr>
<td>T2</td>
</tr>
<tr>
<td>W: ∅</td>
</tr>
<tr>
<td>R: ∅</td>
</tr>
</tbody>
</table>

- Fast user-level faults

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>WRITE Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 2</td>
<td>READ Y</td>
</tr>
</tbody>
</table>

Multi-threaded Application

Software Race Detector

Inter-thread Sharing Monitor
Ideal Hardware Sharing Detector

- Follow read/write sets of threads

  Thread 1
  WRITE Y

  Thread 2
  READ Y

- Fast user-level faults
Ideal Hardware Sharing Detector

- Follow read/write sets of threads

  **Sharing Monitor**
  
  Thread 1
  WRITE Y
  T1
  R: ∅
  W: {Y}

  Thread 2
  READ Y
  T2
  R: ∅
  W: ∅

- Fast user-level faults

Multi-threaded Application

Software Race Detector

Inter-thread Sharing Monitor
Limitations of Existing Hardware

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

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

![Diagram](image.png)
Limitations of Existing Hardware

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

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

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

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

![Diagram showing Multi-threaded Application, Software Race Detector, and Inter-thread Sharing Monitor]
Limitations of Existing Hardware

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

![Diagram showing the relationship between a Multi-threaded Application, Software Race Detector, and Inter-thread Sharing Monitor.]
Limitations of Existing Hardware

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

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

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

- Fast faults
  - Solution: Enable detector for long periods of time
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→R Data Sharing

<table>
<thead>
<tr>
<th>S</th>
<th>M</th>
<th>S</th>
<th>I</th>
</tr>
</thead>
</table>

**Limitations of this method:**
- SMT sharing can’t be counted
- Cache eviction
- Others in paper
Technique 2: Hardware Sharing Detector

- **Hardware Performance Counters**
  - Interrupt on cache coherency events
  - Intel’s HITM event: W→R Data Sharing

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

Execute Instruction
Demand-Driven Analysis on Real HW

Execute Instruction

Analysis Enabled?

YES
Demand-Driven Analysis on Real HW

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

1. Execute Instruction
2. Analysis Enabled?
   - YES: SW Race Detection
   - NO: Sharing Recently?
     - YES: EXECUTE INSTRUCTION
     - NO: EXECUTE INSTRUCTION
Demand-Driven Analysis on Real HW

1. Execute Instruction
   - Analysis Enabled?
     - YES
       - SW Race Detection
     - NO
       - Sharing Recently?
         - YES

Demand-Driven Analysis on Real HW

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

- HITM Interrupt?
  - NO
  - YES

- Analysis Enabled?
  - NO
  - YES

- Execute Instruction

- SW Race Detection

- Disable Analysis

- Sharing Recently?
  - NO
  - YES
Demand-Driven Analysis on Real HW

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

- NO
Demand-Driven Analysis on Real HW

Flowchart:
- **HITM Interrupt?**
  - **YES**: Enable Analysis
  - **NO**: Proceed to next step

- **Execute Instruction**
  - **NO**: Proceed to next step
  - **YES**: Proceed to next step

- **Analysis Enabled?**
  - **NO**: Proceed to next step
  - **YES**: SW Race Detection

- **Sharing Recently?**
  - **NO**: Proceed to next step
  - **YES**: Disable Analysis

- **Disable Analysis**
  - **YES**: Proceed to next step
  - **NO**: Proceed to next step
Demand-Driven Analysis on Real HW

1. **HITM Interrupt?**
   - **NO** → Execute Instruction
   - **YES** → Enable Analysis

2. **Execute Instruction**
   - **NO** → Analysis Enabled?
     - **NO** → Hitm Interrupt?
       - **NO** → Disable Analysis
       - **YES** → Sharing Recently?
         - **NO** → SW Race Detection
         - **YES** → Analysis Enabled?

3. **Analysis Enabled?**
   - **NO** → Hitm Interrupt?
     - **NO** → Disable Analysis
     - **YES** → Sharing Recently?
       - **NO** → SW Race Detection
       - **YES** → Analysis Enabled?
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)

<table>
<thead>
<tr>
<th>Phoenix</th>
<th>PARSEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>histogram</td>
<td>270 ± 15</td>
</tr>
<tr>
<td>kmeans</td>
<td>250 ± 10</td>
</tr>
<tr>
<td>regression</td>
<td>180 ± 5</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>150 ± 2</td>
</tr>
<tr>
<td>string_match</td>
<td>100 ± 1</td>
</tr>
<tr>
<td>word_count</td>
<td>80 ± 1</td>
</tr>
<tr>
<td>GeoMean</td>
<td>70 ± 1</td>
</tr>
<tr>
<td>blackscholes</td>
<td>60 ± 1</td>
</tr>
<tr>
<td>bodytrack</td>
<td>50 ± 1</td>
</tr>
<tr>
<td>facesim</td>
<td>40 ± 1</td>
</tr>
<tr>
<td>ferret</td>
<td>35 ± 1</td>
</tr>
<tr>
<td>fregmine</td>
<td>30 ± 1</td>
</tr>
<tr>
<td>raytrace</td>
<td>25 ± 1</td>
</tr>
<tr>
<td>swapsions</td>
<td>20 ± 1</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>15 ± 1</td>
</tr>
<tr>
<td>vips</td>
<td>10 ± 1</td>
</tr>
<tr>
<td>x264</td>
<td>5 ± 1</td>
</tr>
<tr>
<td>canmeal</td>
<td>2 ± 1</td>
</tr>
<tr>
<td>dedup</td>
<td>1 ± 1</td>
</tr>
<tr>
<td>streamcluster</td>
<td>0 ± 1</td>
</tr>
<tr>
<td>GeoMean</td>
<td>0 ± 1</td>
</tr>
</tbody>
</table>
Performance Increases

![Performance Increases Graph]

- **Phoenix** vs. **PARSEC**

  - Speedup (x) for various benchmarks:
    - 51x
    - Other benchmarks show varying degrees of performance increases.

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

![Graph showing demand-driven analysis speedup (x) for various benchmarks and the GeoMean. The graph includes bars for benchmarks like histogram, kmeans, matrix_multiply, pca, word_count, GeoMean, blackscholes, bodytrack, facesim, ferret, fregm, raytrace, swaptions, fluidanimate, vips, x264, canneal, dedup, streamcluster, and GeoMean. The red box highlights a speedup of 2/4 compared to the GeoMean.]
Demand-Driven Analysis Accuracy

The graph illustrates the demand-driven analysis accuracy for various benchmarks. Each benchmark is represented by a bar, with the speedup (x) indicated at the top. The benchmarks include histogram, kmeans, linear_regression, matrix_multiply, pca, string_match, word_count, GeomMean, blackscholes, bodytrack, facesim, ferret, freqmine, raytrace, swapsions, fluidanimate, vips, x264, canneal, dedup, streamcluster, and GeomMean. The speedup values range from 1/1 to 4/4.
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
Concurrency Bugs Matter NOW

Thread 1
mylen=small

Thread 2
mylen=large

Nov. 2010 OpenSSL Security Flaw

```c
if(ptr == NULL) {
    len=thread_local->mylen;
    ptr=malloc(len);
    memcpy(ptr, data, len);
}
```
Concurrency Bugs Matter **NOW**

**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)
Concurrency Bugs Matter **NOW**

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 \ø
Demand-Driven Analysis Algorithm

1. Thread Executes Instruction
2. Synchro Operation?
   - NO
   - YES: Mark All Data Unshared, Update Lamport Clocks
3. Memory Operation?
   - NO
   - YES: Shared with another thread?
     - NO
     - YES: Write shared?
       - NO
       - YES: Run Software Race Detector
     - YES: Update Lamport Clocks
Demand-Driven Analysis on Real HW
### 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>W→W</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>R→W</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>W→R</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>W→W</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>R→W</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>W→R</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>