AtomTracker: A Comprehensive Approach to Atomic Region Inference and Violation Detection

-Abdullah Muzahid, Norimasa Otsuki, Josep Torrellas
University of Illinois at Urbana-Champaign
Debugging a multithreaded program has a lot in common with medieval torture methods”

-- Random quote found via Google search
Concurrent Bugs

- Multicore era $\Rightarrow$ more parallel programs
  $\Rightarrow$ more concurrency bugs
Concurrent Bugs

- Multicore era => more parallel programs => more concurrency bugs
- Atomicity violation bug
  - A type of concurrency bug
  - Reason: too short critical sections
  - Result: accesses from different threads interleave incorrectly
  - Very frequent, gets relatively less attention
Atomicity Violation Bug

Example

class Point { int x, y; };

<table>
<thead>
<tr>
<th>Thread1</th>
<th>Thread2</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock(l);</td>
<td>lock(l);</td>
</tr>
</tbody>
</table>
| p.x = ... | p.x = ...
| ...       | ...     |
| p.y = ... | p.y = ...
| unlock(l);| unlock(l);
| lock(l);  | lock(l);|
| p.x = ... | p.x = ...
| ...       | ...     |
| p.y = ... | p.y = ...
| unlock(l);| unlock(l);|
## Atomicity Violation Bug Example

A class `Point` with two integer variables `x` and `y` is shown. Two threads, `Thread1` and `Thread2`, access these variables with locks and unlocks. The code snippet demonstrates how atomic operations can be violated when threads concurrently access shared variables without proper synchronization.

```java
class Point { int x, y; }

// Thread 1
lock(l);
p.x = ...            ...
p.y = ...
unlock(l);

// Thread 2
lock(l);
p.x = ...
unlock(l);
lock(l);
p.y = ...
unlock(l);
```

```java
// Thread 1
lock(l);
p.x = ...
unlock(l);

// Thread 2
lock(l);
p.x = ...
unlock(l);
lock(l);
p.y = ...
unlock(l);
```
class Point {
    int x, y;
};

<table>
<thead>
<tr>
<th>Thread1</th>
<th>Thread2</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock(l);</td>
<td>lock(l);</td>
</tr>
<tr>
<td>p.x = ...</td>
<td>lock(l);</td>
</tr>
<tr>
<td></td>
<td>unlock(l);</td>
</tr>
<tr>
<td></td>
<td>p.y = ...</td>
</tr>
<tr>
<td></td>
<td>unlock(l);</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thread2

<table>
<thead>
<tr>
<th>Thread1</th>
<th>Thread2</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock(l);</td>
<td>lock(l);</td>
</tr>
<tr>
<td>p.x = ...</td>
<td>lock(l);</td>
</tr>
<tr>
<td>unlock(l);</td>
<td>unlock(l);</td>
</tr>
<tr>
<td>p.y = ...</td>
<td></td>
</tr>
<tr>
<td>unlock(l);</td>
<td></td>
</tr>
<tr>
<td>lock(l);</td>
<td></td>
</tr>
<tr>
<td>p.y = ...</td>
<td></td>
</tr>
<tr>
<td>unlock(l);</td>
<td></td>
</tr>
</tbody>
</table>

**Atomicity Violation Bug Example**
class Point { int x, y; }

Thread1                  Thread2
lock(l);
p.x = …
…
p.y = …
unlock(l);

Thread1                  Thread2
lock(l);
p.x = …
unlock(l);
lock(l);
p.x = …
…
p.y = …
unlock(l);

Thread1                  Thread2
lock(l);
unlock(l);
lock(l);
unlock(l);
lock(l);
lock(l);
p.y = …
unlock(l);
lock(l);
p.y = …
unlock(l);

• Data race freedom does not imply atomicity bug freedom
State of the Art in Atomicity Violation Detection

- Many proposals
- We are interested in those that require no user annotations
- They are constrained in the types of Atomic Regions (AR):
  - Number of variables
  - Number of instructions
  - Type of code construction (e.g., a function)
State of the Art in Atomicity Violation Detection

- Many proposals
- We are interested in those that require no user annotations
- They are constrained in the types of Atomic Regions (AR):
  - Number of variables
  - Number of instructions
  - Type of code construction (e.g., a function)
- Example: AVIO [Lu06]

\[
\text{Access}(x) \quad \text{Access}(x) \quad \text{Access}(x)
\]

AtomTracker: A Comprehensive Approach ...
Outline

- Motivation
- Contributions
- Main Idea
- Results
- Conclusions
Outline

• Motivation
• Contributions
• Main Idea
• Results
• Conclusions
Contributions
Contributions

- **Novel algorithm** to infer arbitrary atomic regions (AR)
  - Needs no annotation at all
Contributions

• **Novel algorithm** to infer arbitrary atomic regions (AR)
  – Needs no annotation at all
• **Novel algorithm** to detect atomicity violations at runtime
  – A software implementation
  – A hardware implementation with negligible execution overhead
Contributions

• **Novel algorithm** to infer arbitrary atomic regions (AR)
  – Needs no annotation at all

• **Novel algorithm** to detect atomicity violations at runtime
  – A software implementation
  – A hardware implementation with negligible execution overhead

• **First proposal** that works with any AR
  – Any number of variables
  – Any number of instructions
  – Not dependent on code construct
Contributions

• **Novel algorithm** to infer arbitrary atomic regions (AR)
  – Needs no annotation at all
• **Novel algorithm** to detect atomicity violations at runtime
  – A software implementation
  – A hardware implementation with negligible execution overhead
• **First proposal** that works with any AR
  – Any number of variables
  – Any number of instructions
  – Not dependent on code construct
• Detects 8 atomicity violation bugs from real code
Outline

• Motivation
• Contributions
• Main Idea
• Results
• Conclusions
Proposal: AtomTracker
Proposal: AtomTracker

- It has two parts:
Proposal: AtomTracker

• It has two parts:
  • AtomTracker-I: Automatically infers generic ARs
Proposal: AtomTracker

- It has two parts:
  - **AtomTracker-I**: Automatically infers generic ARs
  - **AtomTracker-D**: Automatically detects violations of them at runtime
AtomTracker-I

- Infers ARs without programmer’s annotations
AtomTracker-I

- Infers ARs without programmer’s annotations
- Input:
  - Traces of multiple correct runs
  - Each trace is a total order of memory accesses during one execution
AtomTracker-I

• Infers ARs without programmer’s annotations
• Input:
  – Traces of multiple correct runs
  – Each trace is a total order of memory accesses during one execution
• Approach: greedily try to find the largest possible ARs
Example of How AtomTracker-I Works

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rd X</td>
<td>Wr Y</td>
</tr>
<tr>
<td>Rd Y</td>
<td>Wr X</td>
</tr>
<tr>
<td>Rd X</td>
<td>Rd X</td>
</tr>
<tr>
<td>Wr X</td>
<td>Wr Y</td>
</tr>
<tr>
<td>Wr Y</td>
<td>Rd Y</td>
</tr>
<tr>
<td>Wr Y</td>
<td>Wr Y</td>
</tr>
</tbody>
</table>
Example of How AtomTracker-I Works

AtomTracker: A Comprehensive Approach...
### Example of How AtomTracker-I Works

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Wr Y</td>
</tr>
<tr>
<td>Rd X, Rd Y</td>
<td>Wr X</td>
</tr>
<tr>
<td>Rd X</td>
<td>Rd X</td>
</tr>
<tr>
<td>Wr X</td>
<td>Wr Y</td>
</tr>
<tr>
<td>Wr Y</td>
<td>Rd Y</td>
</tr>
<tr>
<td>Wr Y</td>
<td></td>
</tr>
</tbody>
</table>
Example of How AtomTracker-I Works

AtomTracker: A Comprehensive Approach...
Example of How AtomTracker-I Works

AtomTracker: A Comprehensive Approach ...
Example of How AtomTracker-I Works

\[
\begin{array}{c|c}
T1 & T2 \\
\hline
\text{Rd X, Rd Y} & \text{Wr Y} \\
\text{Rd X} & \text{Wr X} \\
\text{Wr X} & \text{Rd X} \\
\text{Wr Y} & \text{Wr Y} \\
\text{Wr Y} & \text{Rd Y} \\
\end{array}
\]
Example of How AtomTracker-I Works

T1

Rd X, Rd Y

Wr X

Wr Y

T2

Wr Y

Rd X

Rd X, Wr X

Wr Y

Wr Y

Rd Y

Wr Y
Example of How AtomTracker-I Works

AtomTracker: A Comprehensive Approach ...
Example of How AtomTracker-I Works

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rd X, Rd Y</td>
<td>Wr Y</td>
</tr>
<tr>
<td>Wr X</td>
<td></td>
</tr>
<tr>
<td>Rd X</td>
<td></td>
</tr>
<tr>
<td>Wr Y</td>
<td></td>
</tr>
<tr>
<td>Rd X, Wr X, Wr Y</td>
<td></td>
</tr>
<tr>
<td>Rd Y</td>
<td></td>
</tr>
<tr>
<td>Wr Y</td>
<td></td>
</tr>
</tbody>
</table>
Example of How AtomTracker-I Works

AtomTracker: A Comprehensive Approach...
Example of How AtomTracker-I Works

\[ T1 \quad T2 \]

\begin{align*}
\text{Rd X, Rd Y} \\
\text{Wr X} \\
\text{Rd X} \\
\text{Wr Y} \\
\text{Rd X, Wr X, Wr Y} \\
\text{Rd Y} \\
\text{Wr Y}
\end{align*}
Example of How AtomTracker-I Works

AtomTracker: A Comprehensive Approach ...
Example of How AtomTracker-I Works
• Given a trace, find out the largest possible ARs
• As we see more and more traces, the larger ARs will get divided into multiple smaller ARs
• Eventually, we will find ARs close to the actual ones
Illustrative Example of Convergence

Trace 1 → Trace 2 → Trace 3
Illustrative Example of Convergence
Design Decisions of AtomTracker-I

- Ignore synchronization accesses
  - Sync accesses might be incorrect

- Treat critical sections as indivisible group of instructions during the merging process
  - Typically critical sections are supposed to be atomic

- Finish AR at loop iteration boundaries
  - AR usually do not stride loop iteration
Result of Applying AtomTracker-I

- AR boundaries coincide with
  - Synchronizations
  - Data races

- Some AR contain multiple critical sections (CS)
AtomTracker-D

- Takes a program with annotations in the binary
  - AR_enter, AR_exit
- Detects violations of these AR at runtime
- Idea: As two ARs execute concurrently, AtomTracker-D checks if they can be made to appear to execute in sequence
  - If not: atomicity violation

---

No violation if

```
AR1
AR2
OR
```

```
AR2
AR1
```

---

AtomTracker: A Comprehensive Approach ...

Abdullah Muzahid
How AtomTracker-D Works

- Considers each access of the two concurrent ARs in order and checks for conflicts
- Each processor keeps a local flag
  - Tells the order of own AR relative to other AR
  - Values: \{Before (B), After (A), Unordered (U)\}

\[
\begin{array}{ccc}
P1 & \text{Order}_{AR1 \rightarrow AR2} & P2 \\
\begin{array}{l}
\text{WR X} \\
\text{RD X} \\
\text{RD Y} \\
\text{RD X} \\
\end{array} & & \\
\begin{array}{l}
\text{RD X} \\
\text{WR X} \\
\text{WR Y} \\
\end{array}
\end{array}
\]
How AtomTracker-D Works

- Considers each access of the two concurrent ARs in order and checks for conflicts
- Each processor keeps a local flag
  - Tells the order of own AR relative to other AR
  - Values: {Before (B), After (A), Unordered (U)}

\[ \text{P1} \quad \text{P2} \]

<table>
<thead>
<tr>
<th>Order (_{AR1 \rightarrow AR2})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{U})</td>
</tr>
</tbody>
</table>

\[ \text{AR1} \quad \text{WR } X \quad \text{AR2} \]

AtomTracker: A Comprehensive Approach ...
How AtomTracker-D Works

- Considers each access of the two concurrent ARs in order and checks for conflicts
- Each processor keeps a local flag
  - Tells the order of own AR relative to other AR
  - Values: \{Before (B), After (A), Unordered (U)\}

\[
\text{Order}_{\text{AR1} \rightarrow \text{AR2}}
\]

\[
\begin{align*}
P1 & \quad \text{P2} \\
\text{Wr} X & \quad \text{Rd} X \\
\text{AR1} & \quad \text{AR2}
\end{align*}
\]

\[
\begin{align*}
U & \\
U \& B = B
\end{align*}
\]
How AtomTracker-D Works

- Considers each access of the two concurrent ARs in order and checks for conflicts
- Each processor keeps a local flag
  - Tells the order of own AR relative to other AR
  - Values: \{Before (B), After (A), Unordered (U)\}

\[
\begin{align*}
P1 & \quad P2 \\
\text{AR1} & \quad \text{AR2} \\
\text{Wr X} & \quad \text{Rd X} \\
\text{Rd X} &
\end{align*}
\]

\[
\text{Order}_{\text{AR1}\rightarrow\text{AR2}}
\]

- \(U\)
- \(U \& B = B\)
- \(B \& U = B\)
How AtomTracker-D Works

- Considers each access of the two concurrent ARs in order and checks for conflicts.
- Each processor keeps a local flag:
  - Tells the order of own AR relative to other AR.
  - Values: \{Before (B), After (A), Unordered (U)\}

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wr X</td>
<td>Rd X</td>
</tr>
<tr>
<td>Rd X</td>
<td></td>
</tr>
<tr>
<td>Rd Y</td>
<td></td>
</tr>
</tbody>
</table>

**AR1**

**AR2**

\[ \text{Order}_{AR1 \rightarrow AR2} \]

- U
- U & B = B
- B & U = B
- B & U = B
How AtomTracker-D Works

- Considers each access of the two concurrent ARs in order and checks for conflicts
- Each processor keeps a local flag
  - Tells the order of own AR relative to other AR
  - Values: {Before (B), After (A), Unordered (U)}

**Order** \(_{AR1 \rightarrow AR2}\)

- \(U\)
- \(U \& B = B\)
- \(B \& U = B\)
- \(B \& B = B\)

**P1**

- Wr X
- Rd X
- Rd Y
- Wr Y

**P2**

- Rd X

**AR1**

**AR2**
How AtomTracker-D Works

- Considers each access of the two concurrent ARs in order and checks for conflicts
- Each processor keeps a local flag
  - Tells the order of own AR relative to other AR
  - Values: \{Before (B), After (A), Unordered (U)\}

\[
\begin{align*}
\text{P1} & \quad \text{P2} \\
\text{WR X} & \quad \text{Rd X} \\
\text{Rd X} & \quad \text{WR Y} \\
\text{Rd Y} & \quad \text{Rd X}
\end{align*}
\]

\[
\begin{align*}
\text{Order}_{\text{AR1} \rightarrow \text{AR2}} & \quad \text{Violation} \\
\text{U} & \quad \text{No} \\
\text{U \& B} & \quad \text{=> No} \\
\text{B \& U} & \quad \text{=> No} \\
\text{B \& B} & \quad \text{=> No}
\end{align*}
\]
How AtomTracker-D Works

- Considers each access of the two concurrent ARs in order and checks for conflicts
- Each processor keeps a local flag
  - Tells the order of own AR relative to other AR
  - Values: {Before (B), After (A), Unordered (U)}

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>Order(_{AR1\rightarrow AR2})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wr X</td>
<td></td>
<td>U</td>
</tr>
<tr>
<td>Rd X</td>
<td>Rd X</td>
<td>U &amp; B = B</td>
</tr>
<tr>
<td>Rd X</td>
<td></td>
<td>B &amp; U = B</td>
</tr>
<tr>
<td>Rd Y</td>
<td>Wr Y</td>
<td>B &amp; U = B</td>
</tr>
<tr>
<td>Rd Y</td>
<td></td>
<td>B &amp; B = B</td>
</tr>
</tbody>
</table>

AtomTracker: A Comprehensive Approach
How AtomTracker-D Works

- Considers each access of the two concurrent ARs in order and checks for conflicts
- Each processor keeps a local flag
  - Tells the order of own AR relative to other AR
  - Values: {Before (B), After (A), Unordered (U)}

\[
\begin{array}{c}
\text{AR1} \\
\begin{align*}
\text{P1} & : \\
\text{Wr X} & , \\
\text{Rd X} & , \\
\text{Rd Y} & , \\
\end{align*} \\
\begin{align*}
\text{AR2} & : \\
\text{Wr Y} & , \\
\end{align*} \\
\text{P2} & : \\
\text{Rd X} & , \\
\text{B} & , \\
\end{array}
\]

\[\text{Order}_{\text{AR1} \rightarrow \text{AR2}}\]
- \(\text{U}\)
- \(\text{U} \& \text{B} = \text{B}\)
- \(\text{B} \& \text{U} = \text{B}\)
- \(\text{B} \& \text{B} = \text{B}\)
- \(\text{B} \& \text{A} \implies \text{Violation}\)

AtomTracker: A Comprehensive Approach ...
AtomTracker-D Implementation

- **Software:**
  - Use PIN
  - For each access, check a software data structure
- **Hardware:**
  - Leverage the cache coherence protocol messages
  - Negligible execution overhead
AtomTracker-D Hardware Implementation

• Key insight: AtomTracker-D
  – Does not need to see all accesses
  – Only needs to see those that can change Order flag

• What are these accesses?
  – Those that introduce WAR, WAW, RAW deps
    • First one of these induces cache coh msg

• Problem: first access in an AR may be invisible to coherence protocol
  – First read to line in S state or first R/W to line in D state
  – Solution: R and W FirstAccess bits in cache tags
Bus Based Implementation

- Hardware module (AVM) on bus
- Signatures to summarize accesses with little state
  - Detect ordering conflicts of ARs
- Each memory access updates the Order flags in all the processors using signatures
- Suffers from false sharing and false positives
Outline

• Motivation
• Contributions
• Main Idea
• Results
• Conclusions
Evaluation

• 8 real atomicity bugs from
  – Apache,
  – MySql and
  – Mozilla suite.
Evaluation

- Simulated AtomTracker-D hardware with Simics

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multicore Core</td>
<td>4 cores at 4 GHz</td>
</tr>
<tr>
<td></td>
<td>4 issue out-of-order</td>
</tr>
<tr>
<td>L1 cache (private)</td>
<td>32 KB, 4 way, 2 cycle lat</td>
</tr>
<tr>
<td>L2 cache (private)</td>
<td>512 KB, 8 way, 12 cycle lat</td>
</tr>
<tr>
<td>Cache line</td>
<td>64B</td>
</tr>
<tr>
<td>Memory</td>
<td>80 cycle round trip lat</td>
</tr>
<tr>
<td>Network</td>
<td>Bus</td>
</tr>
<tr>
<td>Bus bandwidth</td>
<td>128B/cycle</td>
</tr>
<tr>
<td>Coherence protocol</td>
<td>MESI</td>
</tr>
<tr>
<td>Signatures</td>
<td>2 Kbit</td>
</tr>
</tbody>
</table>
## Bug Description

<table>
<thead>
<tr>
<th>Bug #</th>
<th>Version</th>
<th># Variables involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 1</td>
<td>2.0.48</td>
<td>Single</td>
</tr>
<tr>
<td>Apache 2</td>
<td>2.0.46</td>
<td>Single</td>
</tr>
<tr>
<td>Mozilla 1</td>
<td>0.8</td>
<td>Multiple</td>
</tr>
<tr>
<td>Mozilla 2</td>
<td>0.8</td>
<td>Multiple</td>
</tr>
<tr>
<td>Mozilla 3</td>
<td>0.9</td>
<td>Multiple</td>
</tr>
<tr>
<td>MySQL 1</td>
<td>4.0.12</td>
<td>Single</td>
</tr>
<tr>
<td>MySQL 2</td>
<td>3.23.56</td>
<td>Multiple</td>
</tr>
<tr>
<td>MySQL 3</td>
<td>4.0.16</td>
<td>Multiple</td>
</tr>
</tbody>
</table>
## Bug Detection

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Apache 2</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 2</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
## Bug Detection

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Apache 2</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 2</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
## Bug Detection

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Apache 2</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 2</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Bug Detection

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Apache 2</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 2</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
## Bug Detection

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Apache 2</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 2</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

- Detects both single and multiple variable bugs
## Bug Detection

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Apache 2</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 2</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mozilla 3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>MySQL 3</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

- Detects both single and multiple variable bugs
- Detects all bugs, even those not detected by others
AtomTracker-I Training

- It takes 10 - 40 training runs
## AtomTracker-D Overhead

<table>
<thead>
<tr>
<th>App</th>
<th>Hardware Impl Overhead</th>
<th>Execution Time Increase (%)</th>
<th>Traffic Increase(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 1</td>
<td>0.1</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>Apache 2</td>
<td>0.1</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>Mozilla 1</td>
<td>0.1</td>
<td>5.5</td>
<td></td>
</tr>
<tr>
<td>Mozilla 2</td>
<td>0.5</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td>Mozilla 3</td>
<td>0.1</td>
<td>2.7</td>
<td></td>
</tr>
<tr>
<td>MySQL 1</td>
<td>0.1</td>
<td>4.2</td>
<td></td>
</tr>
<tr>
<td>MySQL 2</td>
<td>0.1</td>
<td>1.5</td>
<td></td>
</tr>
<tr>
<td>MySQL 3</td>
<td>0.3</td>
<td>3.8</td>
<td></td>
</tr>
<tr>
<td>AVG</td>
<td><strong>0.2</strong></td>
<td><strong>3.3</strong></td>
<td></td>
</tr>
</tbody>
</table>

- Negligible execution time and traffic overhead
- Suitable for production runs

AtomTracker: A Comprehensive Approach ...
### AtomTracker-D Overhead

<table>
<thead>
<tr>
<th>App</th>
<th>Software Impl Overhead</th>
<th>Pin Only (X)</th>
<th>Total (X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 1</td>
<td>8.7</td>
<td>80.4</td>
<td></td>
</tr>
<tr>
<td>Apache 2</td>
<td>6.9</td>
<td>74.1</td>
<td></td>
</tr>
<tr>
<td>Mozilla 1</td>
<td>2.9</td>
<td>14.6</td>
<td></td>
</tr>
<tr>
<td>Mozilla 2</td>
<td>1.3</td>
<td>2.1</td>
<td></td>
</tr>
<tr>
<td>Mozilla 3</td>
<td>1.5</td>
<td>1.9</td>
<td></td>
</tr>
<tr>
<td>MySQL 1</td>
<td>6.2</td>
<td>15.9</td>
<td></td>
</tr>
<tr>
<td>MySQL 2</td>
<td>7.5</td>
<td>13.9</td>
<td></td>
</tr>
<tr>
<td>MySQL 3</td>
<td>1.8</td>
<td>2.8</td>
<td></td>
</tr>
<tr>
<td>AVG</td>
<td>4.6</td>
<td>25.7</td>
<td></td>
</tr>
</tbody>
</table>

- Reasonable for testing environment
- Software implementation is improvable
## False Positives (FP)

<table>
<thead>
<tr>
<th>Application</th>
<th>Software Impl</th>
<th>Hardware Impl</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No false sharing &amp; aliasing</td>
<td>Only false sharing, no aliasing</td>
</tr>
<tr>
<td>Apache 1</td>
<td>1.8</td>
<td>2.0</td>
</tr>
<tr>
<td>Apache 2</td>
<td>2.6</td>
<td>10.4</td>
</tr>
<tr>
<td>Mozilla 1</td>
<td>0.4</td>
<td>1.8</td>
</tr>
<tr>
<td>Mozilla 2</td>
<td>0.0</td>
<td>3.0</td>
</tr>
<tr>
<td>Mozilla 3</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>MySQL 1</td>
<td>0.6</td>
<td>12.2</td>
</tr>
<tr>
<td>MySQL 2</td>
<td>0.8</td>
<td>7.6</td>
</tr>
<tr>
<td>MySQL 3</td>
<td>0.2</td>
<td>17.2</td>
</tr>
<tr>
<td>AVG</td>
<td>0.8</td>
<td>6.8</td>
</tr>
</tbody>
</table>
 Outline

- Motivation
- Contributions
- Main Idea
- Results
- Conclusions
Conclusions

• AtomTracker – A novel technique for atomicity violation detection
  – AtomTracker-I automatically infers ARs
  – AtomTracker-D automatically detects violations at run-time
    • Suitable to implement in hardware using signatures
• First proposal to handle arbitrary AR
• Detected 8 atomicity violations in real world code
• Hardware implementation induces negligible exec overhead → production runs
AtomTracker: A Comprehensive Approach to Atomic Region Inference and Violation Detection

- Abdullah Muzahid, Norimasa Otsuki, Josep Torrellas
  University of Illinois at Urbana-Champaign
## Atomic Regions

<table>
<thead>
<tr>
<th>Applications</th>
<th># LOC</th>
<th># Shared Var</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 1</td>
<td>44.3</td>
<td>62.4</td>
</tr>
<tr>
<td>Apache 2</td>
<td>76.4</td>
<td>125.6</td>
</tr>
<tr>
<td>Mozilla 1</td>
<td>62.2</td>
<td>82.9</td>
</tr>
<tr>
<td>Mozilla 2</td>
<td>83.8</td>
<td>197.1</td>
</tr>
<tr>
<td>Mozilla 3</td>
<td>46.4</td>
<td>102.1</td>
</tr>
<tr>
<td>MySQL 1</td>
<td>43.9</td>
<td>230.4</td>
</tr>
<tr>
<td>MySQL 2</td>
<td>44.3</td>
<td>82.4</td>
</tr>
<tr>
<td>MySQL 3</td>
<td>17.0</td>
<td>31.8</td>
</tr>
<tr>
<td>AVG</td>
<td>52.3</td>
<td>114.3</td>
</tr>
<tr>
<td>SP2 kernels</td>
<td>4.6</td>
<td>215.5</td>
</tr>
<tr>
<td>SP2 apps</td>
<td>4.7</td>
<td>28.2</td>
</tr>
</tbody>
</table>

AtomTracker: A Comprehensive Approach ...
Effectiveness of Prepasses

<table>
<thead>
<tr>
<th>Microbenchmarks</th>
<th>ATI (%)</th>
<th>ATI-CS (%)</th>
<th>ATI-LP (%)</th>
<th>ATI-LP – CS (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkedList (17)</td>
<td>100</td>
<td>58.8</td>
<td>94.1</td>
<td>52.9</td>
</tr>
<tr>
<td>ProdCons (4)</td>
<td>100</td>
<td>100</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>FFT (14)</td>
<td>100</td>
<td>78.6</td>
<td>100</td>
<td>78.6</td>
</tr>
<tr>
<td>AVG</td>
<td>100</td>
<td>79.1</td>
<td>89.7</td>
<td>68.8</td>
</tr>
</tbody>
</table>

- Both passes are essential
MySQL 2 Bug

Thread 1

1. lock(l);
2. t->rows = 0;
3. lock(b);
4. binlog .write("DELETE");
5. unlock(b);

Thread 2

1. lock(l);
2. unlock(l);
3. lock(b);
4. t->rows ++;
5. lock(b);
6. binlog .write("INSERT");
7. unlock(b);

DELETE & INSERT recorded in correct order

DELETE & INSERT recorded in wrong order
Problem With Nesting

thread 1
lock(l);
... = y;
lock(m);
x = ...;
unlock(m);

lock(m);
x = ...;
unlock(m);
y = ...;
unlock(l);

thread 2

lock(m);

... = y;
lock(m);
x = ...;
unlock(m);

lock(m);
x = ...;
unlock(m);
y = ...;
unlock(l);