A Consistency Checker for Memory Subsystem Traces

Matthew Naylor, Simon Moore, Alan Mujumdar

Email: matthew.naylor@cl.cam.ac.uk
Problem

Verify that the **memory subsystem** in a shared-memory multiprocessor implements a well-defined consistency model.

This is a prerequisite for the **correct execution** of concurrent programs on such architectures.
Our approach

**Black-box** specification-based testing:

1. Feed auto-generated requests to mem subsystem
2. Record a **trace** of all requests and responses
3. Check that trace satisfies consistency model
Attractions of black-box approach

**Generic:** can be applied to a wide range of implementations and coherence protocols.

**Easy to apply:** no modifications are required to the design under test.
Drawback of black-box approach

Checking traces is an \textbf{NP-complete} problem \cite{gibbons1994}.

\textbf{Corollary}: larger traces involving more cores are more likely to contain bugs yet less likely to be checkable in reasonable time.
State of the art

**TSOtool** [Manovit, 2006] is a conformance checker for the TSO consistency model.

It can handle **large traces**, on the order of millions of memory operations and hundreds of cores.

Achieved through powerful inference rules and careful algorithm design.
BUT...

Many modern memory subsystem implementations are more relaxed than TSO.

And TSOtool is a “proprietary product of Sun Microsystems”.
Example: Limitations of TSO

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>*data := 1</td>
<td>*flag := 1</td>
</tr>
<tr>
<td>*flag := 1</td>
<td>*data := 0</td>
</tr>
</tbody>
</table>

Forbidden under TSO, but observable if:

- L1 cache is non-blocking, e.g. Rocket Chip, where first load is a miss & second is a hit.
- Or, coherence protocol is lazy, e.g. BERI, where second load is a stale hit.
Our main contributions

- **Generalisation** of TSOtool’s algorithm to support a wider range of consistency models.

- An open-source checker for memory subsystem traces called **Axe**.

- **Experiences** of applying Axe to open-source SoCs BERI and Rocket Chip.
Part II: Axe Consistency Checker
What is Axe?

**Input**: a memory subsystem trace

<table>
<thead>
<tr>
<th>Thread id</th>
<th>Action</th>
<th>Value</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>M[0] := 1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>sync</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>{ M[1] == 0; M[1] := 1 }</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>M[1] == 1</td>
<td>100:110</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>M[0] == 0</td>
<td>115</td>
<td></td>
</tr>
</tbody>
</table>

**Output**: Does trace satisfy SC, TSO, PSO, WMO or POW model? If not, emit smallest subset of trace that fails.
SPARC models

- **SC** prohibits reordering.
- **TSO** can reorder S → L, simulating store buffer.
- **WMO** can additionally reorder S → S, L → L, and L → S (provided addresses differ).
## Algorithm demo

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>M[0] := 1</td>
<td>M[0] == 1</td>
<td>M[0] := 2</td>
</tr>
<tr>
<td>M[0] == 2</td>
<td>M[0] := 3</td>
<td></td>
</tr>
</tbody>
</table>
Add thread-local edges

Thread 0

\[ M[0] := 1 \]

Thread 1

\[ M[0] == 1 \]

\[ M[0] == 2 \]

Thread 2

\[ M[0] := 2 \]

\[ M[0] := 3 \]
Add reads-from edges

Thread 0  Thread 1  Thread 2

M[0] := 1  M[0] == 1  M[0] := 2
M[0] == 2  M[0] := 3
Delete a root, add reads-before edges

Thread 0

\[ M[0] := 1 \]

Thread 1

\[ M[0] == 1 \]
\[ M[0] == 2 \]

Thread 2

\[ M[0] := 2 \]
\[ M[0] := 3 \]
Violation: cycle detected!

Thread 0

\[ M[0] := 1 \]

\[ M[0] == 1 \]

\[ M[0] := 2 \]

\[ M[0] == 2 \]

Thread 1

\[ M[0] == 1 \]

\[ M[0] == 2 \]

Thread 2

\[ M[0] := 2 \]

\[ M[0] := 3 \]
Backtrack, delete a root, add reads-before edges

Thread 0

\[ M[0] := 1 \]

Thread 1

\[ M[0] == 1 \]
\[ M[0] := 2 \]
\[ M[0] == 2 \]

Thread 2

\[ M[0] := 2 \]
\[ M[0] := 3 \]
Delete a root

Thread 0

\[ M[0] := 1 \]

Thread 1

\[ M[0] := 1 \]

\[ M[0] == 2 \]

Thread 2

\[ M[0] := 2 \]

\[ M[0] := 3 \]
Delete root, add reads-before edges

Thread 0

\[ M[0] := 1 \]

Thread 1

\[ M[0] := 2 \]

Thread 2

\[ M[0] := 3 \]
Delete root

Thread 0

\[ M[0] := 1 \]

Thread 1

\[ M[0] := 1 \]

\[ M[0] := 2 \]

Thread 2

\[ M[0] := 2 \]

\[ M[0] := 3 \]
Delete root

Thread 0

\[ M[0] := 1 \]

Thread 1

\[ M[0] := 2 \]

Thread 2

\[ M[0] := 3 \]

Empty graph -- trace is valid!
Lesson

- Easy to encounter **backtracking** behaviour during topological sort.
- Routine backtracking is catastrophic for checking even small traces.
- In response, TSOtool uses **inference rules**.
TSOtool’s inference rules

Rule 1

\[ M[x] := v \]

\[ M[x] := w \]

\[ M[x] == v \]

Rule 2

\[ M[x] := v \]

\[ M[x] := w \]

\[ M[x] == w \]
Apply Rule 2

Picking a root is now deterministic
Efficient graph representation

- During checking, adding an edge to the graph is a very common operation.

- **Problem**: need to quickly determine whether any added edge introduces a cycle.

- Sounds like maintenance of an $O(N^3)$ transitive closure, disastrous for large $N$.
SC graph representation

- Under **SC**, operations on the same thread are **totally ordered**.

- For each node, we need only maintain the nearest successor on each thread.

- Complexity: $O(N*T)$
TSO graph representation

- Under **TSO**, loads on the same thread are totally ordered. Likewise for stores.

- For each node, we need only maintain the nearest load & store successor on each thread.

- Complexity: $O(2^N\times T)$
WMO graph representation

- Under **WMO**, loads from same address on same thread are **totally ordered**. Likewise for stores.
- For each node, maintain the nearest load & store successor on each thread for each address.
- Complexity: $O(2*N*T*A)$
- Still much better than $O(N^3)$: $T$ and $A$ are small.
Axe performance evaluation (WMO)

Averaged over a range of traces (576 in total):

Checking time grows **linearly** with trace size.
Trace shrinking

**Problem**: It’s hard to determine *why* a large trace is invalid just by staring at it.

**Solution**: A trace shrinking procedure.

Given a trace that violates a model, it searches for the *smallest subset* of the trace that still violates the model.
Part III: Applications
We replaced the core with a random traffic generator that logs all requests & responses, yielding a **random trace**.
Rocket Chip coherence bug

260-element counterexample, after shrinking:

0: \( M[2] := 46 \) @ 497:

1: \( M[2] == 46 \) @ 280:513

1: \( M[2] := 61 \) @ 729:

1: \( M[2] == 46 \) @ 854:979

Only write of 46 in trace Write of 61 dropped

Identified as “race condition” by Rocket Chip devs.
Rocket Chip atomics bug

After shrinking:

1:  M[3] := 31

Not atomic

Bug occurs when a store-conditional is issued before a load-reserve response is received.
BERI barrier bug

After shrinking:

1: M[39028] := 76
1: M[39028] := 79  # Set data
1: sync
1: M[2761] := 83  # Set flag
0: M[2761] == 83  # See flag
0: sync
0: M[39028] == 76  # See stale data

This bug only observable after generating cancelled loads and stores in traffic generator.
Summary & conclusions

- We have **generalised** a state-of-the-art checker to a wider range of consistency models through our open-source tool Axe.

- This enabled us to test BERI & Rocket Chip, uncovering several serious bugs, **concisely reported** using our trace shrinking procedure.

- Time complexity now dependent on number of distinct addresses in trace, but **still performs well**.