DACOTA: Post-silicon Validation of the Memory Subsystem in Multi-core Designs

Presenter: Bo Zhang
Yulin Shi
Outline

• Motivation & Goal
• Solution - DACOTA overview
• Technical Insights
• Experimental Evaluation
• Conclusion
• Discussion points
Motivation & Goal

Question:
What is the difference between Verification and Validation?

Jason Stinson, 2006, Pre-Si Verification for Post-Si Validation [PPT],
http://www.slideshare.net/DVClub/presi-verification-for-postsi-validation/4
Motivation & Goal

Problem:
• Bottleneck of pre-silicon verification
• Increasing complexity – Multiprocessors
• Shrinking production schedule
• More 10% of errors from memory subsystem

Current Solution:
• Post-silicon functional Validation (high speed)
• Logic Analyzers, On-chip Assertion, and Scan Chain

However... limited by internal observability.
Solution – DACOTA Overview

A new validation solution for memory operation ordering in Multi-core designs.

Advantage:

• High coverage <- executing in chip
• Debugging support <- activity log
• Near-zero area overhead <- disable DACOTA
• Small performance impact <- 26% slowdown only during Validation
Solution – DACOTA Overview

Activity Logging
• Access vector
• Core activity log

Policy Validation Algorithm
• Access log aggregation
• Graph construction
• Graph analysis
Technical Insights – Activity Logging

• Access Vector

a. \[
\begin{array}{cccc}
\text{Data} & 0 & 0 & 0 \\
\text{Core0} & \text{Core1} & \text{Core2} & \text{Counter}
\end{array}
\]

b. \[
\begin{array}{cccc}
\text{Data} & 1 & 3 & 2 \\
\text{Core0} & \text{Core1} & \text{Core2} & \text{Counter}
\end{array}
\]

c. \[
\begin{array}{cccc}
\text{Data} & 3 & 0 & 0 \\
\text{Core0} & \text{Core1} & \text{Core2} & \text{Counter}
\end{array}
\]

d. \[
\begin{array}{cccc}
\text{Data} & 3 & 2 & 0 \\
\text{Core0} & \text{Core1} & \text{Core2} & \text{Counter}
\end{array}
\]

• Core Activity Log

Pending:

<table>
<thead>
<tr>
<th>Access</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load C</td>
<td>2</td>
</tr>
<tr>
<td>Load D</td>
<td>3</td>
</tr>
</tbody>
</table>

Index Table

\[
\begin{array}{|c|c|}
\hline
\text{Access} & \text{Entry} \\
\hline
\text{Load C} & 2 \\
\text{Load D} & 3 \\
\hline
\end{array}
\]

Queue full

Index counter

Program order

Activity log

- Acc. vector of A
- Acc. vector of B
- Acc. vector of C
- Acc. vector of D
Technical Insights – Activity Logging

Activity Logging Example

- Vector format: $c_0 | c_1 \text{cnt}$
- $c_0$: entry for Core0
- $c_1$: entry for Core1
- $\text{cnt}$: counter number
Technical Insights – Validation Algorithm

Access Log Aggregation

• Invoked when log resources exhausted
• All cores stop execution and complete pending memory operation
• Freeze data portions of cache
• Activity logs transferred to un-cacheable memory
Technical Insights – Validation Algorithm

Graph Construction

• Build a directed graph, show order of operations
• Vertices: memory operation
• Edge: ordering constraints
• Preliminary check
• Check coherence invariant
  • Sequential Consistency
  • Total Store Ordering (TSO)
  • Processor Consistency
  • Other Consistency models
Technical Insights – Validation Algorithm

Graph Analysis

Dacota searches the graphs for loops:
• Employing Depth First Search (DFS) algorithm
• Exploits cores of CMP to run the algorithm
• Reconfigure previous log storage cache as regular cache
Error Detection Example

Coherence conflict Example
Error Detection Example

Sequential Consistency Violation Example
Experimental Evaluation-Error coverage

Dacota is capable of quickly finding complex coherence and consistency bugs.

<table>
<thead>
<tr>
<th>Bug name</th>
<th>Description of the error</th>
<th>Avg. cycles to expose</th>
</tr>
</thead>
<tbody>
<tr>
<td>shared_store</td>
<td>store to a shared line may not invalidate other caches</td>
<td>0.252M</td>
</tr>
<tr>
<td>invisible_store</td>
<td>store message may not reach all cores</td>
<td>1.32M</td>
</tr>
<tr>
<td>store_alloc_1</td>
<td>store allocation in any core may not occur properly</td>
<td>1.93M</td>
</tr>
<tr>
<td>store_alloc_2</td>
<td>store allocation in a single core may not occur properly</td>
<td>2.27M</td>
</tr>
<tr>
<td>reorder_1</td>
<td>invalid store reordering (all cores)</td>
<td>1.38M</td>
</tr>
<tr>
<td>reorder_2</td>
<td>invalid store reordering (by a single core)</td>
<td>2.82M</td>
</tr>
<tr>
<td>reorder_3</td>
<td>invalid store reordering (to a single address)</td>
<td>2.87M</td>
</tr>
<tr>
<td>reorder_4</td>
<td>invalid store reordering (to a single address by a single core)</td>
<td>5.61M</td>
</tr>
</tbody>
</table>
Experimental Evaluation—Performance

- Average performance overhead for SPLASH2 is 26%.
- For random benchmark, the overhead is higher.
- Not a problem. Since Dacota can be disabled upon shipment.

CMP specs
- 16 cores, each with 16 entry load/store buffer
- 128KB L1 cache
- 4MB L2 cache
- Activity log size: 256 entries
Experimental Evaluation - Performance

- For some benchmarks, computation overhead decreases as the activity log size grows.
- For some overhead time exhibit a local minimum at medium log size.
  - Growing complexity of consistency graph results in more analyze time.

Figure 11. Dacota computation overhead. a. Overhead for SPLASH2 benchmarks (4 individual benchmarks and average of all 10 benchmarks is shown) b. Overhead for directed random tests (4 individual tests and average of all 10 tests is shown).
Experimental Evaluation-Performance

• Communication remains nearly constant despite different queue size and workload
Experimental Evaluation - Performance

- Time between checks tend to grow with increasing activity log size.
Experimental Evaluation – Area overhead

• Dacota reuses existing hardware, like cores and storage
• Small area overhead (less than 0.01%)
• Three orders of magnitude smaller than previous solutions
Related work comparison

3 major approaches to memory subsystem verification

- Pre-silicon verification
  - Easy to check in design state space
  - Slow speed simulation
  - Hard to achieve high coverage

- Runtime solution
  - Provide effective protection against soft errors
  - Large hardware overhead

- Post-silicon validation
  - Effective in exposing bugs
  - Small performance overhead (only in validation)
  - Small silicon area overhead
  - Bug catching ability depends on quality of the stimulus
Debate Question

• 1, Before analysis, the activity log data is transferred to un-cacheable memory, is it valid to assume that the access to that part of memory is error-free?

• 2, Can DACOTA method also be implemented during Pre-silicon verification? Why or why not?

• 3, Is it valid to assume that load hit accesses to local caches are always serviced correctly?
Reference


2, Jason Stinson, 2006, *Pre-Si Verificaiton for Post-Si Validation [PPT]*, http://www.slideshare.net/DVClub/presi-verification-for-postsi-validation/4