Correctly Implementing Value Prediction in Microprocessors that Support Multithreading or Multiprocessing

Milo M.K. Martin, Daniel J. Sorin, Harold W. Cain, Mark D. Hill, and Mikko H. Lipasti

Computer Sciences Department
Department of Electrical and Computer Engineering
University of Wisconsin—Madison

(C) 2001 Daniel Sorin

Big Picture

- Naïve value prediction can break concurrent systems
- Microprocessors incorporate concurrency
  - Multithreading (SMT)
  - Multiprocessing (SMP, CMP)
  - Coherent I/O
- Correctness defined by memory consistency model
  - Comparing predicted value to actual value not always OK
  - Different issues for different models
- Violations can occur in practice
- Solutions exist for detecting violations

Outline

- The Issues
  - Value prediction
  - Memory consistency models
- The Problem
- Value Prediction and Sequential Consistency
- Value Prediction and Relaxed Consistency Models
- Conclusions

Value Prediction

- Predict the value of an instruction
  - Speculatively execute with this value
  - Later verify that prediction was correct
- Example: Value predict a load that misses in cache
  - Execute instructions dependent on value-predicted load
  - Verify the predicted value when the load data arrives
- Without concurrency: simple verification is OK
  - Compare actual value to predicted
- Value prediction literature has ignored concurrency

Memory Consistency Models

- Correctness defined by consistency model
- Rules about legal orderings of reads and writes
  - E.g., do all processors observe writes in the same order?
- Example: Sequential consistency (SC)
  - Simplest memory model
  - System appears to be multitasking uniprocessor

Outline

- The Issues
- The Problem
  - Informal example
  - Linked list code example
- Value Prediction and Sequential Consistency
- Value Prediction and Relaxed Consistency Models
- Conclusions

Appearance of one memory operation at a time
Informal Example of Problem, part 1

- Student #2 predicts grades are on bulletin board B
- Based on prediction, assumes score is 60

<table>
<thead>
<tr>
<th>Grades for Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student ID</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Informal Example of Problem, part 2

- Professor now posts actual grades for this class
  - Student #2 actually got a score of 80
- Announces to students that grades are on board B

<table>
<thead>
<tr>
<th>Grades for Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student ID</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Informal Example of Problem, part 3

- Student #2 sees prof’s announcement and says, “I made the right prediction (bulletin board B), and my score is 60”!
- Actually, Student #2’s score is 80
- What went wrong here?
  - Intuition: predicted value from future
  - Problem is concurrency
  - Interaction between student and professor
  - Just like multiple threads, processors, or devices
    - E.g., SMT, SMP, CMP

Linked List Example of Problem (initial state)

- Linked list with single writer and single reader
- No synchronization (e.g., locks) needed

Linked List Example of Problem (Writer)

- Writer sets up node B and inserts it into list

Linked List Example of Problem (Reader)

- Reader cache misses on head and value predicts head=B
- Cache hits on B.data and reads 60
- Later “verifies” prediction of B. Is this execution legal?

Code For Writer Thread

\[
\begin{align*}
W1: & \text{ store mem} [B.data] \leftarrow 80 \\
W2: & \text{ load reg0 } \leftarrow \text{ mem} [\text{Head}] \\
W3: & \text{ store mem} [B.next] \leftarrow \text{ reg0} \\
W4: & \text{ store mem} [\text{Head}] \leftarrow B
\end{align*}
\]

Code For Reader Thread

\[
\begin{align*}
R1: & \text{ load reg1 } \leftarrow \text{ mem} [\text{Head}] = B \\
R2: & \text{ load reg2 } \leftarrow \text{ mem} [\text{reg1}] = 60
\end{align*}
\]
Why This Execution Violates SC

- Sequential Consistency
  - Simplest memory consistency model
  - Must exist total order of all operations
  - Total order must respect program order at each processor

- Our example execution has a cycle
  - No total order exists

Trying to Find a Total Order

- What orderings are enforced in this example?

<table>
<thead>
<tr>
<th>Code For Writer Thread</th>
<th>Code For Reader Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1: store mem[B.data] &lt;- 80</td>
<td>R1: load reg1 &lt;- mem[Head]</td>
</tr>
<tr>
<td>W2: load reg0 &lt;- mem[Head]</td>
<td>R2: load reg2 &lt;- mem[reg1]</td>
</tr>
</tbody>
</table>

Program Order

- Must enforce program order

<table>
<thead>
<tr>
<th>Code For Writer Thread</th>
<th>Code For Reader Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1: store mem[B.data] &lt;- 80</td>
<td>R1: load reg1 &lt;- mem[Head]</td>
</tr>
<tr>
<td>W2: load reg0 &lt;- mem[Head]</td>
<td>R2: load reg2 &lt;- mem[reg1]</td>
</tr>
</tbody>
</table>

Data Order

- If we predict that R1 returns the value B, we can violate SC

<table>
<thead>
<tr>
<th>Code For Writer Thread</th>
<th>Code For Reader Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2: load reg0 &lt;- mem[Head]</td>
<td>R2: load reg2 &lt;- mem[reg1] = 60</td>
</tr>
</tbody>
</table>

Outline

- The Issues
- The Problem
- Value Prediction and Sequential Consistency
  - Why the problem exists
  - How to fix it
- Value Prediction and Relaxed Consistency Models
- Conclusions

Value Prediction and Sequential Consistency

- Key: value prediction reorders dependent operations
  - Specifically, read-to-read data dependence order
  - Execute dependent operations out of program order
- Applies to almost all consistency models
  - Models that enforce data dependence order
- Must detect when this happens and recover
- Similar to other optimizations that complicate SC
How to Fix SC Implementations

• Address-based detection of violations
  – Student watches board B between prediction and verification
  – Like existing techniques for out-of-order SC processors
  – Track stores from other threads
  – If address matches speculative load, possible violation

• Value-based detection of violations
  – Student checks grade again at verification
  – Also an existing idea
  – Replay all speculative instructions at commit
  – Can be done with dynamic verification (e.g., DIVA)

Outline

• The Issues
• The Problem
• Value Prediction and Sequential Consistency
  • Value Prediction and Relaxed Consistency Models
    – Relaxed consistency models
    – Value prediction and processor consistency (PC)
    – Value prediction and weakly ordered models
• Conclusions

Relaxed Consistency Models

• Relax some orderings between reads and writes
• Allows HW/SW optimizations
• Software must add memory barriers to get ordering
• Intuition: should make value prediction easier
• Our intuition is wrong …

Processor Consistency

• Just like SC, but relaxes order from write to read
• Optimization: allows for FIFO store queue
• Examples of PC models:
  – SPARC Total Order
  – IA-32
• Bad news
  – Same VP issues as for SC
  – Intuition: VP breaks read-to-read dependence order
  – Relaxing write-to-read order doesn’t change issues
• Good news
  – Same solutions as for SC

Weakly Ordered Consistency Models

• Relax orderings unless memory barrier between
• Examples:
  – SPARC RMO
  – IA-64
  – PowerPC
  – Alpha
• Subtle point that affects value prediction
  – Does model enforce data dependence order?

Models that Enforce Data Dependence

Examples: SPARC RMO, PowerPC, and IA-64

<table>
<thead>
<tr>
<th>Code For Writer Thread</th>
<th>Code For Reader Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1: store mem[B.data] ← 80</td>
<td>R1: load reg1 ← mem[Head]</td>
</tr>
<tr>
<td>W2: load reg0 ← mem[Head]</td>
<td>R2: load reg2 ← mem[reg1]</td>
</tr>
<tr>
<td>W3: store mem[B.next] ← reg0</td>
<td></td>
</tr>
<tr>
<td>W3b: Memory Barrier</td>
<td>Memory barrier orders W4 after W1, W2, W3</td>
</tr>
<tr>
<td>W4: store mem[Head] ← B</td>
<td></td>
</tr>
</tbody>
</table>
Violating Consistency Model

- Simple value prediction can break RMO, PPC, IA-64
- How? By relaxing dependence order between reads
- Same issues as for SC and PC

Solutions to Problem

1. Don’t enforce dependence order (add memory barriers)
   - Changes architecture
   - Breaks backward compatibility
   - Not practical
2. Enforce SC or PC
   - Potential performance loss
3. More efficient solutions possible

Models that Don’t Enforce Data Dependence

- Example: Alpha
- Requires extra memory barrier (between R1 & R2)

<table>
<thead>
<tr>
<th>Code For Writer Thread</th>
<th>Code For Reader Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2: load reg0 &lt;- mem[Head]</td>
<td>R1b: Memory Barrier</td>
</tr>
<tr>
<td>W3b: Memory Barrier</td>
<td></td>
</tr>
<tr>
<td>W4: store mem[Head] &lt;- B</td>
<td></td>
</tr>
</tbody>
</table>

Issues in Not Enforcing Data Dependence

- Works correctly with value prediction
  - No detection mechanism necessary
  - Do not need to add any more memory barriers for VP
- Additional memory barriers
  - Non-intuitive locations
  - Added burden on programmer

Summary of Memory Model Issues

Could this Problem Happen in Practice?

- Theoretically, value prediction can break consistency
- Could it happen in practice?

- Experiment:
  - Ran multithreaded workloads on SimOS
  - Looked for code sequences that could violate model

- Result: sequences occurred that could violate model
Conclusions

• Naïve value prediction can violate consistency
• Subtle issues for each class of memory model

• Solutions for SC & PC require detection mechanism
  – Use existing mechanisms for enhancing SC performance

• Solutions for more relaxed memory models
  – Enforce stronger model