The Programming Interface

• WO and RC require synchronized programs

• All synchronization operations must be labeled and visible to the hardware
  – Easy (easier!) if synchronization library used
  – Must provide language support for arbitrary Ld/St synchronization
    (event notification, e.g., flag)

• Program written for weaker model OK on stricter
  – E.g., SC is a valid implementation of TSO, WO, or RC
A Mind-Bending Example

• What happens when memory consistency interacts with value prediction?
• Hint: it’s not obvious!
Informal Example of Problem, part 1

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

Grades for Class

<table>
<thead>
<tr>
<th>Student ID</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
</tr>
<tr>
<td>3</td>
<td>85</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>Student ID</th>
<th>Score 1</th>
<th>Score 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>85</td>
<td>70</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

Initial state of list

Uninitialized node

(C) 2010 Daniel J. Sorin from Adve, Falsafi, Hill, Lebeck, Reinhardt, Singh
• Writer sets up node B and inserts it into list

```
> Code For Writer Thread

W1: store mem[B.data] <- 80
W2: load reg0 <- mem[Head]
W3: store mem[B.next] <- reg0
W4: store mem[Head] <- B
```

(C) 2010 Daniel J. Sorin from Adve, Falsafi, Hill, Lebeck, Reinhardt, Singh
Linked List Example of Problem (Reader)

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

\begin{itemize}
  \item \textbf{Predict head} = B
  \end{itemize}

\begin{figure}
\centering
\begin{tikzpicture}
\node[draw, fill=blue!20] (head) at (0,0) {?};
\node[draw, fill=blue!20] (A-data) at (0,-1) {42 \hspace{1cm} null};
\node[draw, fill=blue!20] (A-next) at (0,-2) {null};
\node[draw, fill=blue!20] (B-data) at (1,-1) {60 \hspace{1cm} null};
\node[draw, fill=blue!20] (B-next) at (1,-2) {B.data \hspace{1cm} B.next};
\node at (0,1) {\textbf{Code For Reader Thread}};
\node at (1,1) {\textbf{R1: load reg1 } \leftarrow \text{ mem[Head] } = \text{ B} \hspace{3cm} \text{R2: load reg2 } \leftarrow \text{ mem[reg1] } = \text{ 60}};
\end{tikzpicture}
\end{figure}

(C) 2010 Daniel J. Sorin from Adve, Falsafi, Hill, Lebeck, Reinhardt, Singh
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?

Code For Writer Thread

W1: store mem[B.data] <- 80
W2: load reg0 <- mem[Head]
W3: store mem[B.next] <- reg0
W4: store mem[Head] <- B

Code For Reader Thread

R1: load reg1 <- mem[Head]
R2: load reg2 <- mem[reg1]
Program Order

• Must enforce program order

Code For Writer Thread

W1: store mem[B.data] <- 80
W2: load reg0 <- mem[Head]
W3: store mem[B.next] <- reg0
W4: store mem[Head] <- B

Code For Reader Thread

R1: load reg1 <- mem[Head]
R2: load reg2 <- mem[reg1]
If we predict that R1 returns the value B, we can violate SC.

**Code For Writer Thread**

W1: store mem[B.data] <- 80
W2: load reg0 <- mem[Head]
W3: store mem[B.next] <- reg0
W4: store mem[Head] <- B

**Code For Reader Thread**

R1: load reg1 <- mem[Head] = B
R2: load reg2 <- mem[reg1] = 60
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 w/Value Pred

- Two options from “Two Techniques for …”
  - Both adapted from ICPP ’91 paper on course website (optional reading)
  - Originally developed for out-of-order SC cores
- (1) 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
- (2) 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

• Difference Between Coherence and Consistency

• Sequential Consistency

• Relaxed Memory Consistency Models

• Consistency Optimizations

• Synchronization Optimizations
Consistency is an Abstraction

- We only have to implement a system that preserves the illusion of the specified consistency model
  - Like OOO processor preserves illusion of in-order execution

- We can speculate and recover if speculation leads to violation of consistency model
  - E.g., MIPS R10000 processor speculates on consistency

- **Scheurich’s Optimization** (for directory protocols)
  - Immediately acknowledge incoming Invalidation (to Shared block)
  - Yet continue to read this block (i.e., pretend Inv didn’t arrive yet)
  - Must invalidate before asking mem system for upgrade to any block
  - Optimization does NOT even violate SC

(C) 2010 Daniel J. Sorin from Adve, Falsafi, Hill, Lebeck, Reinhardt, Singh
/* initially A=B=0, P1 owns A, P2 shares B */

P1

issue GETX B

ST B=1

ST A=1

P2

LD B=0

Ack Inv B

LD B=0 /* Legal! */
/\* initially A=B=0, P1 owns A, P2 shares B */

**P1**
- issue GETX B
- ST B=1
- ST A=1
- send A=1

**P2**
- LD B=0
- Ack Inv B
- LD B=0 /* Legal! */
- issue GETS A
- LD A=1
- LD B=0 /* Illegal */
Is SC+ILP=RC?

- PRESENTATION