Beyond Sequential Consistency: Relaxed Memory Models

Arvind
Computer Science and Artificial Intelligence Lab
M.I.T.

Based on the material prepared by Arvind and Krste Asanovic
Beyond Sequential Consistency: Relaxed Memory Models
Sequential Consistency

- In-order instruction execution
- Atomic loads and stores

SC is easy to understand but architects and compiler writers want to violate it for performance
Memory Model Issues

Architectural optimizations that are correct for uniprocessors, often violate sequential consistency and result in a new memory model for multiprocessors
Example 1: Store Buffers

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store(flag_1,1); r_1 := Load(flag_2);</td>
<td>Store(flag_2,1); r_2 := Load(flag_1);</td>
</tr>
</tbody>
</table>

**Question:** Is it possible that \( r_1 = 0 \) and \( r_2 = 0 \)?

- **Sequential consistency:** No

- Suppose Loads can bypass stores in the store buffer: Yes!

**Total Store Order (TSO):**

- IBM 370, Sparc’s TSO memory model

Initially, all memory locations contain zeros

November 21, 2005
Example 2: Short-circuiting

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store(flag₁, 1);</td>
<td>Store(flag₂, 1);</td>
</tr>
<tr>
<td>( r₃ := \text{Load}(\text{flag₁}) );</td>
<td>( r₄ := \text{Load}(\text{flag₂}) );</td>
</tr>
<tr>
<td>( r₁ := \text{Load}(\text{flag₂}) );</td>
<td>( r₂ := \text{Load}(\text{flag₁}) );</td>
</tr>
</tbody>
</table>

Question: Do extra Loads have any effect?
- **Sequential consistency:** No
- **Suppose Load-Store short-circuiting is permitted in the store buffer**
  - No effect in Sparc’s TSO model
  - A Load acts as a barrier on other loads in IBM 370
Example 3: Non-FIFO Store buffers

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store(a,1);</td>
<td>$r_1 := \text{Load}(\text{flag})$;</td>
</tr>
<tr>
<td>Store(flag,1);</td>
<td>$r_2 := \text{Load}(a)$;</td>
</tr>
</tbody>
</table>

Question: *Is it possible that $r_1=1$ but $r_2=0$?*

- **Sequential consistency:** No
- **With non-FIFO store buffers:** Yes

Sparc’s PSO memory model
Example 4: Non-Blocking Caches

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store(a,1);</td>
<td>$r_1 := \text{Load(flag)}$;</td>
</tr>
<tr>
<td>Store(flag,1);</td>
<td>$r_2 := \text{Load(a)}$;</td>
</tr>
</tbody>
</table>

**Question:** Is it possible that $r_1 = 1$ but $r_2 = 0$?
- **Sequential consistency:** No
- **Assuming stores are ordered:** Yes because Loads can be reordered

Sparc’s RMO, PowerPC’s WO, Alpha

November 21, 2005
Example 5: Register Renaming

Initially both $r_1$ and $r_2$ contain 1.

Question: Is it possible that $r_1=0$ but $r_2=0$?

- Sequential consistency: No
- Register renaming: Yes because it removes anti-dependencies
Example 6: Speculative Execution

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store(a,1);</td>
<td>r₁ := Load(flag);</td>
</tr>
<tr>
<td>Store(flag,1);</td>
<td>Jz(r₁,L);</td>
</tr>
<tr>
<td></td>
<td>r₂ := Load(a);</td>
</tr>
</tbody>
</table>

Question: Is it possible that \( r₁ = 1 \) but \( r₂ = 0 \)?

- **Sequential consistency:** No
- **With speculative loads:** Yes even if the stores are ordered
**Example 7: Store Atomicity**

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
<th>Process 3</th>
<th>Process 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store(a,1);</td>
<td>Store(a,2);</td>
<td>r₁ := Load(a);</td>
<td>r₃ := Load(a);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r₂ := Load(a);</td>
<td>r₄ := Load(a);</td>
</tr>
</tbody>
</table>

**Question:** Is it possible that $r₁=1$ and $r₂=2$ but $r₃=2$ and $r₄=1$?

- **Sequential consistency:** No
- **Even if Loads on a processor are ordered, the different ordering of stores can be observed if the Store operation is not atomic.**
Example 8: Causality

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Process 2</th>
<th>Process 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store(flag₁,1);</td>
<td>r₁ := Load(flag₁);</td>
<td>r₂ := Load(flag₂);</td>
</tr>
<tr>
<td>Store(flag₂,1);</td>
<td>r₂ := Load(flag₁);</td>
<td>r₃ := Load(flag₁);</td>
</tr>
</tbody>
</table>

**Question:** Is it possible that \( r₁ = 1 \) and \( r₂ = 1 \) but \( r₃ = 0 \)?

- **Sequential consistency:** No
Five-minute break to stretch your legs
Weaker Memory Models & Memory Fence Instructions

- Architectures with weaker memory models provide memory fence instructions to prevent the permitted reorderings of loads and stores

\[
\text{Store}(a_1, v); \\
\text{Fence}_{\text{wr}} \\
\text{Load}(a_2); \\
\]

The Load and Store can be reordered if \( a_1 \neq a_2 \). Insertion of \( \text{Fence}_{\text{wr}} \) will disallow this reordering

Similarly:

\[
\text{Fence}_{\text{rr}}; \quad \text{Fence}_{\text{rw}}; \quad \text{Fence}_{\text{ww}}; \\
\]

SUN’s Sparc: MEMBAR; MEMBARRR; MEMBARRW; MEMBARWR; MEMBARWW

PowerPC: Sync; EIEIO

November 21, 2005
Enforcing SC using Fences

**Processor 1**
Store(a,10);
Store(flag,1);

**Processor 2**
L: \( r_1 = \text{Load}(\text{flag}); \)
\( \text{Jz}(r_1, \text{L}); \)
\( r_2 = \text{Load}(a); \)

**Processor 1**
Store(a,10);
Fence_{ww};
Store(flag,1);

**Processor 2**
L: \( r_1 = \text{Load}(\text{flag}); \)
\( \text{Jz}(r_1, \text{L}); \)
Fence_{rr};
\( r_2 = \text{Load}(a); \)

Weak ordering

November 21, 2005
Weaker (Relaxed) Memory Models

- Alpha, Sparc, PowerPC, ...
- TSO, PSO, RMO, ...
- Store is globally performed
- Write-buffers
- SMP, DSM

• Hard to understand and remember
• Unstable - Modèle de l’année
Backlash in the architecture community

• Abandon weaker memory models in favor of SC by employing aggressive “speculative execution” tricks.
  - all modern microprocessors have some ability to execute instructions speculatively, i.e., ability to kill instructions if something goes wrong (e.g. branch prediction)
  - treat all loads and stores that are executed out of order as speculative and kill them if a signal is received from some other processor indicating that SC is about to be violated.
Aggressive SC Implementations

*Loads can go out of order*

<table>
<thead>
<tr>
<th>Processor 1</th>
<th>Processor 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>miss</em></td>
<td></td>
</tr>
<tr>
<td>$r_1 = \text{Load}(\text{flag})$</td>
<td>$\text{Store}(a,10)$</td>
</tr>
<tr>
<td><strong>hit</strong></td>
<td></td>
</tr>
<tr>
<td>$r_2 = \text{Load}(a)$</td>
<td></td>
</tr>
</tbody>
</table>

*kill Load*(a) *and the subsequent instructions if Store*(a,10) happens before Load*(flag) completes*

- Still not as efficient as weaker memory mode
- Scalable for Distributed Shared Memory systems?
Properly Synchronized Programs

- Very few programmers do programming that relies on SC; instead higher-level synchronization primitives are used
  - locks, semaphores, monitors, atomic transactions
- A “properly synchronized program” is one where each shared writable variable is protected (say, by a lock) so that there is no race in updating the variable.
  - There is still race to get the lock
  - There is no way to check if a program is properly synchronized
- For properly synchronized programs, instruction reordering does not matter as long as updated values are committed before leaving a locked region.
Release Consistency

- Only care about inter-processor memory ordering at thread synchronization points, not in between.
- Can treat all synchronization instructions as the only ordering points.

... Acquire(lock) // All following loads get most recent written values
... Read and write shared data..
Release(lock) // All preceding writes are globally visible before
    // lock is freed.
...