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

Processor 1	*Processor 2*
Store(a,10);	L: r_1 = Load(flag);
Store(flag,1);	Jz(r_1,L);
	r_2 = Load(a);

initially flag = 0

- 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

Process 1	*Process 2*
Store($flag_1$,1);	Store($flag_2$,1);
r_1 := Load($flag_2$);	r_2 := Load($flag_1$);

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

Example 2: Short-circuiting

Process 1	Process 2
Store($flag_1$,1);	Store($flag_2$,1);
r_3 := Load($flag_1$);	r_4 := Load($flag_2$);
r_1 := Load($flag_2$);	r_2 := Load($flag_1$);

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

Process 1	*Process 2*
Store(a,1);	r_1 := Load(flag);
Store(flag,1);	r_2 := Load(a);

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

Process 1	Process 2
Store(a,1);	r_1 := Load(flag);
Store(flag,1);	r_2 := Load(a);

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

Example 5: Register Renaming

Process 1	*Process 2*
Store(flag$_1$,r$_1$);	Store(flag$_2$,r$_2$);
r$_1$:= Load(flag$_2$);	r$_2$:= Load(flag$_1$);

Register renaming will eliminate this edge

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

Process 1		*Process 2*
Store(a,1);	L:	r_1 := Load(flag);
Store(flag,1);		Jz(r_1,L);
		r_2 := Load(a);

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

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

Example 7: Store Atomicity

Process 1	*Process 2*	*Process 3*	*Process 4*
Store(a,1);	Store(a,2);	r_1 := Load(a);	r_3 := Load(a);
		r_2 := Load(a);	r_4 := Load(a);

Question: Is it possible that r_1=1 and r_2=2 but r_3=2 and r_4=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

Process 1	*Process 2*	*Process 3*
Store($flag_1$,1);	r_1 := Load($flag_1$);	r_2 := Load($flag_2$);
	Store($flag_2$,1);	r_3 := Load($flag_1$);

Question: Is it possible that r_1=1 and r_2=1 but r_3=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

Store(a_1, v);
$Fence_{wr}$
Load(a_2);

The Load and Store can be reordered if a_1 =/= a_2. Insertion of $Fence_{wr}$ will disallow this reordering

Similarly: $Fence_{rr}$; $Fence_{rw}$; $Fence_{ww}$;

SUN's Sparc: MEMBAR;
MEMBARRR; MEMBARRW; MEMBARWR; MEMBARWW
PowerPC: Sync; EIEIO

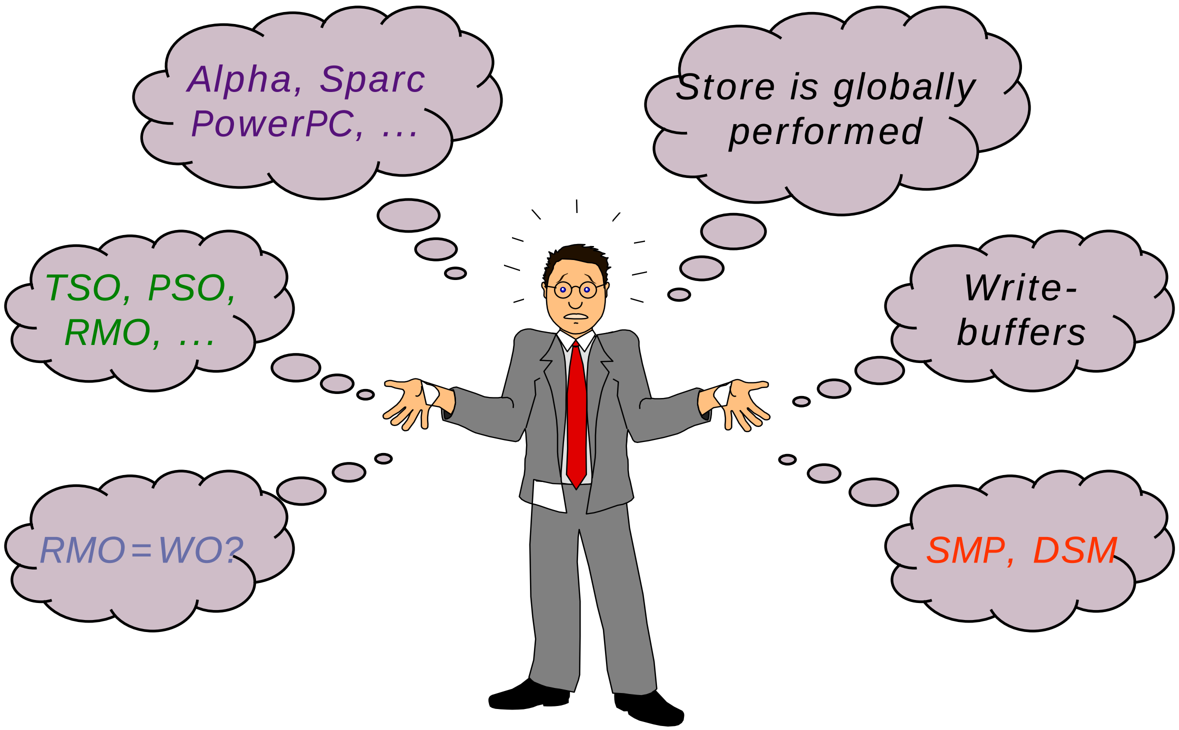
Enforcing SC using Fences

Processor 1	*Processor 2*
Store(a,10);	L: r_1 = Load(flag);
Store(flag,1);	Jz(r_1,L);
	r_2 = Load(a);

Processor 1	*Processor 2*
Store(a,10);	L: r_1 = Load(flag);
$Fence_{ww}$;	Jz(r_1,L);
Store(flag,1);	$Fence_{rr}$;
	r_2 = Load(a);

Weak ordering

Weaker (Relaxed) Memory Models



- 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

	Processor 1	*Processor 2*
miss	r_1 = Load(flag);	Store(a,10);
hit	r_2 = Load(a);	

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.
...
```