Lecture 11: Consistency Models

- Topics: sequential consistency, hw and hw/sw optimizations

Coherence Vs. Consistency

- Recall that coherence guarantees (i) that a write will eventually be seen by other processors, and (ii) write serialization (all processors see writes to the same location in the same order)

- The consistency model defines the ordering of writes and reads to different memory locations – the hardware guarantees a certain consistency model and the programmer attempts to write correct programs with those assumptions

Example Programs

Initially, A = B = 0

```
  P1                        P2
A = 1                     B = 1
if (B == 0)               if (A == 0)
   critical section          critical section
```

```
  P1                  P2
Data = 2000       while (Head == 0)
Head = 1             { }
                  … = Data
```

Initially, A = B = 0

```
 P1            P2              P3
A = 1
          if (A == 1)
             B = 1
                          if (B == 1)
                            register = A
```

Sequential Consistency

P1	P2
Instr-a	Instr-A
Instr-b	Instr-B
Instr-c	Instr-C
Instr-d	Instr-D
…	…

We assume:

- Within a program, program order is preserved
- Each instruction executes atomically
- Instructions from different threads can be interleaved arbitrarily

Valid executions:

abAcBCDdeE… or ABCDEFabGc… or abcAdBe… or
aAbBcCdDeE… or …..

Sequential Consistency

- Programmers assume SC; makes it much easier to reason about program behavior

- Hardware innovations can disrupt the SC model

- For example, if we assume write buffers, or out-of-order execution, or if we drop ACKS in the coherence protocol, the previous programs yield unexpected outputs

Consistency Example - I

- Consider a multiprocessor with bus-based snooping cache coherence and a write buffer between CPU and cache

```
Initially A = B = 0
  P1                  P2
A ← 1               B ← 1
 …                    …
if (B == 0)         if (A == 0)
  Crit.Section        Crit.Section
```

The programmer expected the above code to implement a lock – because of write buffering, both processors can enter the critical section

The consistency model lets the programmer know what assumptions they can make about the hardware's reordering capabilities

Consistency Example - 2

P1	P2
Data = 2000	while (Head == 0) { }
Head = 1	… = Data

Sequential consistency requires program order

-- the write to Data has to complete before the write to Head can begin

-- the read of Head has to complete before the read of Data can begin

Consistency Example - 3

Initially, A = B = 0

P1	P2	P3
A = 1		
	if (A == 1)	
	B = 1	
		if (B == 1)
		register = A

Sequential consistency can be had if a process makes sure that everyone has seen an update before that value is read – else, write atomicity is violated

Sequential Consistency

- A multiprocessor is sequentially consistent if the result of the execution is achieveable by maintaining program order within a processor and interleaving accesses by different processors in an arbitrary fashion

- The multiprocessors in the previous examples are not sequentially consistent

- Can implement sequential consistency by requiring the following: program order, write serialization, everyone has seen an update before a value is read – very intuitive for the programmer, but extremely slow

HW Performance Optimizations

- Program order is a major constraint – the following try to get around this constraint without violating seq. consistency
  - if a write has been stalled, prefetch the block in exclusive state to reduce traffic when the write happens
  - allow out-of-order reads with the facility to rollback if the ROB detects a violation (detected by re-executing the read later)

Relaxed Consistency Models (HW/SW)

- We want an intuitive programming model (such as sequential consistency) and we want high performance

- We care about data races and re-ordering constraints for some parts of the program and not for others – hence, we will relax some of the constraints for sequential consistency for most of the program, but enforce them for specific portions of the code

- Fence instructions are special instructions that require all previous memory accesses to complete before proceeding (sequential consistency)

Fences

P1	P2
{	{
Region of code with no races	Region of code with no races
}	}
Fence	Fence
Acquire_lock	Acquire_lock
Fence	Fence
{	{
Racy code	Racy code
}	}
Fence	Fence
Release_lock	Release_lock
Fence	Fence

Potential Relaxations

- Program Order: (all refer to *different* memory locations)
  - Write to Read program order
  - Write to Write program order
  - Read to Read and Read to Write program orders
- Write Atomicity: (refers to *same* memory location)
  - Read others' write early
- Write Atomicity and Program Order:
  - Read own write early

Relaxations

Relaxation	W → R Order	W → W Order	R →RW Order	Rd others' Wr early	Rd own Wr early
IBM 370	X				
TSO	X				X
PC	X			X	X
SC					X

- IBM 370: a read can complete before an earlier write to a different address, but a read cannot return the value of a write unless all processors have seen the write
- SPARC V8 Total Store Ordering (TSO): a read can complete before an earlier write to a different address, but a read cannot return the value of a write by another processor unless all processors have seen the write (it returns the value of own write before others see it)
- Processor Consistency (PC): a read can complete before an earlier write (by any processor to any memory location) has been made visible to all

Performance Comparison

- Taken from Gharachorloo, Gupta, Hennessy, ASPLOS'91
- Studies three benchmark programs and three different architectures:
  - MP3D: 3-D particle simulator
  - LU: LU-decomposition for dense matrices
  - PTHOR: logic simulator

  - LFC: aggressive; lockup-free caches, write buffer with bypassing
  - RDBYP: only write buffer with bypassing
  - BASIC: no write buffer, no lockup-free caches

Performance Comparison

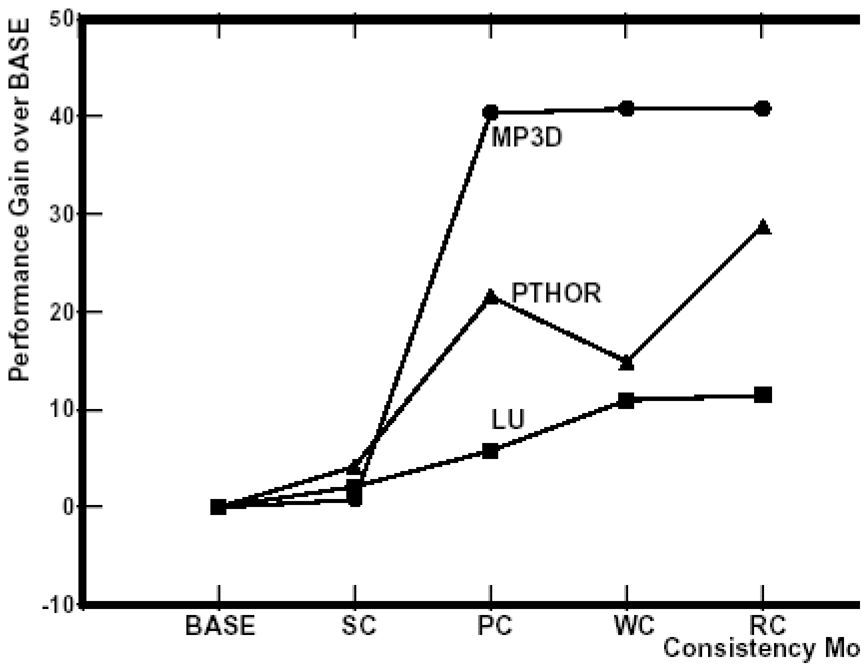

Figure 3: Relative performance of models on LFC

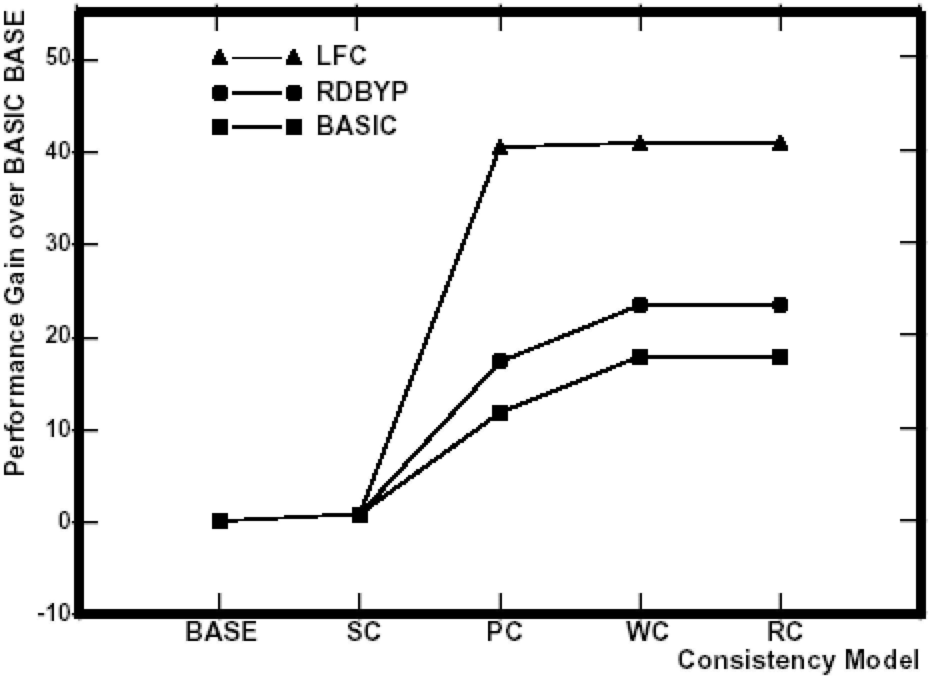

Figure 7: Performance of MP3D under LFC, RDBYP, and BASIC implementations.

Summary

- Sequential Consistency restricts performance (even more when memory and network latencies increase relative to processor speeds)

- Relaxed memory models relax different combinations of the five constraints for SC

- Most commercial systems are not sequentially consistent and rely on the programmer to insert appropriate fence instructions to provide the illusion of SC

Title

- Bullet