Memory Consistency

Suvinay Subramanian

6.823 Spring 2016

Memory Consistency Model

A memory (consistency) model specifies the order in which memory accesses performed by one thread become visible to other threads in the program.

» Contract between the hardware and software

» Loosely speaking, it specifies:

- Set of legal values a load can return
- Set of legal final memory states for a program

Who Cares About Memory Consistency?

» Programmers:

- A framework for writing parallel programs
- Simple reasoning
- Ability to express as much concurrency as possible

» Compiler / Language Designers:

- Allow as many compiler optimizations as possible
- Allow as much implementation flexibility
- Behavior of “bad” programs (?)

Who Cares About Memory Consistency?

» Computer Architects:

- Allow as many hardware optimizations as possible
- Minimize hardware overheads
- Implementation simplicity

Ordering of Memory Operations

» This is not an unfamiliar problem. In a uniprocessor, the memory model is “sequential order”.
  - Hardware (appears to) execute loads and stores in program order

» Out-of-order preserves the same semantics
  - Challenge: Memory dependences → Load-Store Queues

Ordering of Memory Operations

» In a multiprocessor, multiple threads running simultaneously.

» Notion of ”time” / “order” within a given thread and across threads needs to be maintained by hardware in accordance to expectations of software.

Ordering of Memory Operations

» Why is this important in shared memory programming?

- A task may have to proceed "after" a different task has reached a certain point in its code.
- A task may have to ensure no other task is using a resource while it uses it.

» Such communication of "order" often involves multiple addresses.

Sequential Consistency

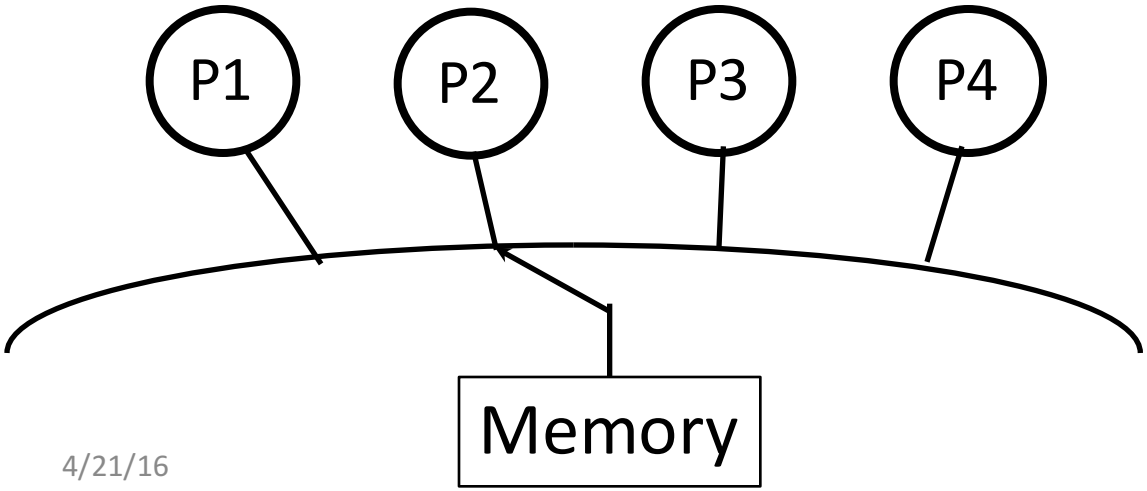
" A system is *sequentially consistent* if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in the order specified by the program".

Leslie Lamport

Sequential Consistency

Two aspects:

1. Maintaining program order among operations from individual processors.
2. Maintaining single sequential order among operations from all processors.



The order of operations observed need not be the same as the physical time the operations were issued.

-> Logical order is what is important.

Violations of Sequential Consistency

» Store—store reordering
  - Non-FIFO store buffers
» Store—Load reordering
  - Store—load bypassing
  - OoO execution
» Register renaming, speculation

Processor Optimizations

» Non-blocking caches

Cache optimization

» Unordered interconnect

Network optimization

» Others…

Store Buffers

Dekker's Algorithm

Initial: `flag1, flag2 = 0`

Processor 1 (P1)

```
flag1 = 1
if (flag2 == 0)
  <critical section>
```

Processor 2 (P2)

```
flag2 = 1
if (flag1 == 0)
  <critical section>
```

Q: Can load of flag1 and flag2 be 0?

In a SC system? No

Loads can go ahead of stores? Yes

Also discussed in L17-11

Overlapping Writes / Unordered Network

Processor 1 (P1)

```
data = 2000
flag = 1
```

Processor 2 (P2)

```
while (flag == 0);
r1 = data
```

Q: What is the value of `r1` in a SC system? 2000

Overlapping Writes / Unordered Network

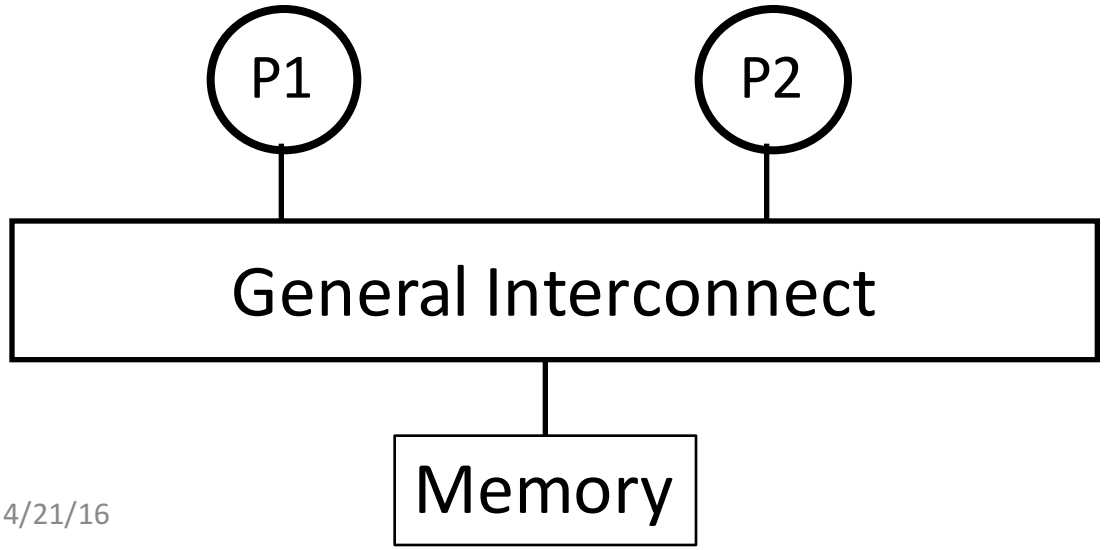
Processor 1 (P1)

```
data = 2000
flag = 1
```

Processor 2 (P2)

```
while (flag == 0);
r1 = data
```

Q: What is the value of `r1` in a SC system? 2000



Q: What if write of `flag` "overtakes" write of `data` ?

Non-blocking Reads / Unordered Network

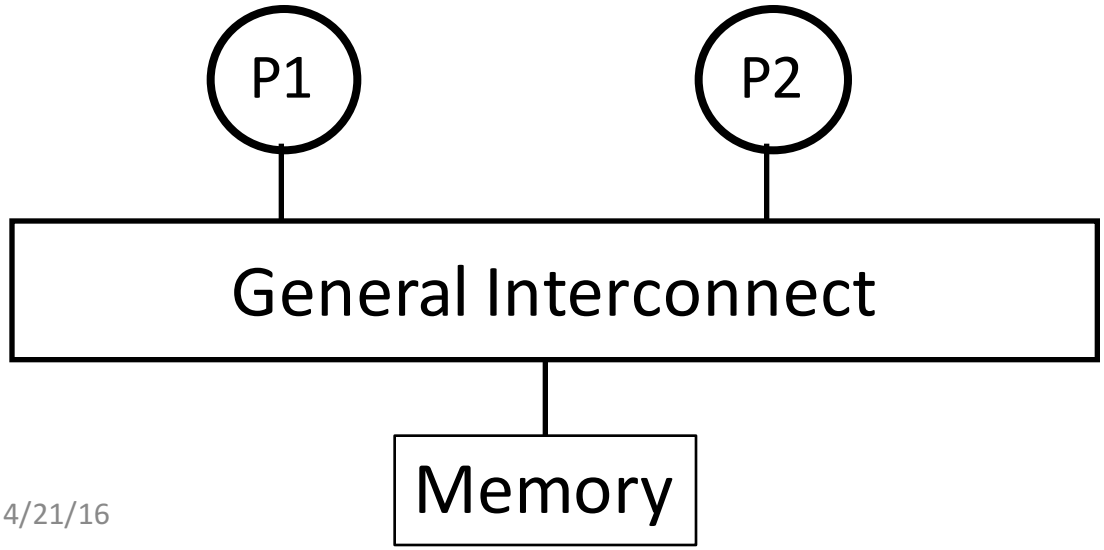
Processor 1 (P1)

```
data = 2000
flag = 1
```

Processor 2 (P2)

```
while (flag == 0);
r1 = data
```

Q: What is the value of `r1` in a SC system? 2000



Q: What if reads are non-blocking?

Relaxed Memory Models

Relax along two axes:

1. Program order requirement
2. Atomicity requirement

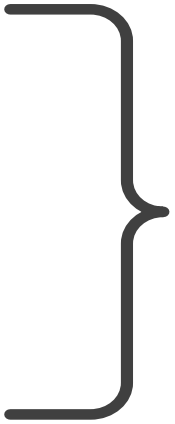
Several models: TSO, Power, Alpha etc.

Relax one or more of the following orderings:

- Load → Load
- Load → Store
- Store → Store
- Store → Load

Total Store Order (TSO)

1. Load → Load
2. Load → Store
3. Store → Store
4. Store → Load



All required for SC

TSO relaxes the last requirement.

What optimization does this allow?

Store buffers

Total Store Order (TSO)

Initial: x, y = 0

Processor 1 (P1)	Processor 2 (P2)
`x = NEW`	`y = NEW`
`r1 = y`	`r2 = x`

Q: Is (`r1`, `r2`) = (0, 0) possible in a SC system? No

Q: Is (`r1`, `r2`) = (0, 0) possible in a TSO system? Yes!

Memory Fences

» Idea: Not all accesses need to be “strictly” ordered. Programmer identifies regions which need (not) be ordered.

»Primitives that prevent otherwise permitted re-orderings of loads and stores

» Different flavors on different systems:

- Sparc: MEMBAR
- x86: LFENCE, SFENCE, MFENCE

Memory Fences

Initial: `flag1`, `flag2` = 0

Processor 1 (P1)

```
flag1 = 1
if (flag2 == 0)
  <critical section>
```

Processor 2 (P2)

```
flag2 = 1
if (flag1 == 0)
  <critical section>
```

Where should you add a fence in a TSO system?

Test Your Understanding

1. Memory consistency model is only relevant to systems with multiple copies of share data (eg. through caching). True/False?

2. Memory consistency model can be defined solely by specifying the behavior of the processor (or only the memory system). True/False?

Test Your Understanding

3. The memory consistency model dictates the legal-orderings of coherence transactions. True/False?

Coherence and Consistency

» Coherence: Concerned with single address.
Consistency: Concerns multiple addresses.

» Consistency: Part of specification of hardware (contract between hardware and software).
Coherence: Transparent to software. Performance optimization. But often used as a mechanism to maintain consistency.

» Studied in a decoupled fashion, but often interact causing non-trivial bugs.

- Eg: Peekaboo problem

Problem 4.11

Processor 1 (P1)	Processor 2 (P2)	Processor 3 (P3)
1.1: ST(A),1	2.1: ST(B),1	3.1: ST(C),1
1.2: LD Rc, (C)	2.2: LD Ra, (A)	3.2: LD Rb, (B)

Q: In a SC system, can {Ra, Rb, Rc} be:

{0, 0, 0}:

{0, 1, 0}:

Problem 4.11

Processor 1 (P1)	Processor 2 (P2)	Processor 3 (P3)
1.1: ST(A),1	2.1: ST(B),1	3.1: ST(C),1
1.2: LD Rc, (C)	2.2: LD Ra, (A)	3.2: LD Rb, (B)

Q: Insert mem fences to prevent non-SC states in a WO system.

Summary

» Memory Consistency:
Contract between hardware and software. Defines valid memory orderings.

» Tradeoff between correctness, ease of programming, complexity, performance

» Relaxed memory models
- Allow certain reorderings
- Use fences to disallow reorderings as required