Memory Consistency Models

Adapted from slides by Sarita V. Adve and Kourosh Gharachorloo
And slides from Gabriel Tanase

Memory Consistency Models

- Memory Consistency (definition)
- Sequential Consistency
- Processor Consistency
- Weak Consistency
- Release Consistency
  - Early Release Consistency
  - Lazy Release Consistency
- Entry Consistency

Memory Consistency

Definition: A memory consistency model for a shared address space specifies constraints on the order in which memory operations must appear to be performed (i.e. to become visible to the processors) with respect to one another.

```
P1
A=1
flag=1

P2
while(flag == 0);
print A;
```

(Culler, Singh, Gupta)
Sequential Consistency

Sequential Consistency (Lamport) “A multiprocessor 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 occur in this sequence in the order specified by its program.”

Implicit Memory Model

- Sequential consistency (SC) [Lamport ’76]
  - Result of an execution appears as if all operations executed in some sequential order
  - Memory operations of each process in program order

Architectures without Caches: example

Initially Flag1 = Flag2 = 0
P1
Flag1 = 1
if (Flag2 == 0)
critical section
P2
Flag2 = 1
if (Flag1 == 0)
critical section

At runtime:
P1
(Operation, Location, Value)
Write, Flag1, 1
Read, Flag2, 0
P2
(Operation, Location, Value)
Write, Flag2, 1
Read, Flag1, 0?
Architectures without Caches: example

P1 (Operation, Location, Value)
Write, Flag1, 1
Read, Flag2, 0

P2 (Operation, Location, Value)
Write, Flag2, 1
Read, Flag1, 0

Can happen if
Reorder write followed by read in h/w or compiler
Allocate copies of Flag1 or Flag2 in registers
Write buffers with read bypassing

Optimization by use of writer buffer is safe on conventional uniprocessor, but it can violate SC in multiprocessor system.

Architectures With Caches

• Cache Coherence and Sequential Consistency (SC)
  – Cache Coherence
    • A write is visible to all processors
    • Serialization of writes to the same location
    – SC
    • Serialization of writes to all locations
    • Operations appear to execute in program order
  • SC implies Cache Coherence:
    A memory consistency model as the policy that places an early and late bound on when a new value can be propagated by invalidating or updating
  • Atomicity for writes
    • Serialize write can avoid the violation of SC
    • Ordering of updates/invalidates between source and destination is preserved by network
    • Or delay an update/invalidate from being sent out until any updates or invalidates from previous write are acknowledged

Sequential Consistency

1. Every process issues memory operations in program order.
2. After a write operation is issued, the issuing process waits for the write to complete before issuing its next operation.
3. After a read operation is issued, the issuing process waits for the read to complete, and for the write whose value is being returned by the read to complete, before issuing its next operation. That is, if the write whose value is being returned has performed with respect to this processor (as it must have if its value is being returned) then the processor should wait until the write has performed with respect to all processors.

Is SC enough?

SC constrains all memory operations:
Simple model for reasoning about parallel programs
But, intuitively reasonable reordering of memory operations in a uniprocessor may violate sequential consistency model in multiprocessor
Modern microprocessors reorder operations all the time to obtain performance (write buffers, overlapped writes, non-blocking reads…).
How do we reconcile sequential consistency model with the demands of performance?
Processor Consistency

- Writes from the same processor should be observed in program order; The order in which the writes from two processors occur (as observed by themselves or a third processor need NOT be identical (Gharachorloo & al.)

1. Before a read is allowed to perform with respect to any other processor, all previous reads must be performed and
2. Before a write is allowed to performed with respect to any other processor all previous accesses (reads and writes) must be performed

- The above conditions relax sequential consistency by allowing reads following a write to bypass the write.

Weak Consistency

- Distinguish Ordinary shared accesses v. Synchronization accesses
- Conditions for weak consistency
  - Before an ordinary read/write access is allowed to perform with respect to any other processor, all previous synchronization accesses must be performed and
  - Before a synchronization access is allowed to performed with respect to any other processor, all previous ordinary read/write accesses must be performed and
- Synchronization accesses are sequentially consistent.

Release Consistency

- Categorization of shared memory accesses
  - Shared access
    - Competing
    - non-competing
  - Synchronization
    - non-synchronization
  - Acquire
  - Release

RC: Properly-Labeled Programs

- Two ways of labelling
  - Parallelizing compilers
  - Programming methodology
Conditions for Release Consistency

- Before an ordinary read or write access is allowed to perform with respect to any other processor, all previous acquire accesses must be performed and
- Before a release access is allowed to perform with respect to any other processor, all previous ordinary read and writes accesses must be performed, and
- Acquire accesses are sequentially consistent with other acquire accesses; same for release access.

Comparison of the four models

Performance Potential

Lazy release consistency for Software DSM
Lazy Release Consistency

Lazy versus Eager

Lazy release consistency

- “happened-before-1” partial order
- Write notice propagation
- Multiple writer protocols
  - Modify different parts of a page concurrently
  - False sharing
  - Reduces the amount of messages
  - Invalidate vs. update(on the acquiring processor)
Other Consistency Models

Entry consistency (Bershad et. Al.)
memory becomes consistent up entry to programmer defined critical section

Scope consistency (Iftode et. Al.)
Any modification during a consistency scope session become visible to processes that enter sessions of that scope
Can have multiple related scope sessions

Delayed Consistency

Delayed protocols

Figure 1: System structure with Store buffers and coherence update buffers

Figure 5: Causal diagram for Stores in a Delayed protocol.
Update based cache protocols


2. Lazy release consistency for software distributed shared memory; Pete Keleher, Alan Cox, Willy Zwaenepoel

3. The Midway distributed shared memory; Brian Bershad & al.

4. Scope Consistency: A bridge between release consistency and entry consistency; L. Iftode, J.P. Singh, K. Li

5. Parallel computer architecture (chapter 5 and 9); David Culler, J.P. Singh, A. Gupta