Memory Consistency Models

Rules that:

- specifies the constraints on the order in which memory operations appear to execute wrt one another
- enables programmers to reason about correctness of operations
- **fence**: program points that memory operations cannot cross
  - **write fence**:
    - all writes that occur before the fence must complete before the fence is executed
    - no write after the fence can begin before the fence has been executed
  - **read fence**: the same, but for reads
  - **memory fence**:
    - serves as both a read fence & a write fence
    - enforces ordering between processors
- synchronization can act as a fence

Memory Consistency Models

Sequential

- model to the programmer
- memory accesses on a processor must appear to be executed in order & to completion
- memory operations from all processors are interleaved in some global order
- if the implementation follows the programmer model exactly, performance suffers
  - all reads (writes) are read (write) fences
  - synchronize (release/acquire) between shared accesses
  - for example,
    - no prefetching
    - no write buffers
    - no write-back caches without cache-to-cache transfers
    - no out of order execution

Memory Consistency Models

Weaker or relaxed consistency models

- relax to improve performance
  - later operation can complete first
  - hides memory latencies
- still preserves a simple programming model for synchronized programs

Memory Consistency Models

The vehicle:

- define a model by which single-processor read/write orderings it preserves
- the orderings:
  - R $\rightarrow$ R
  - R $\rightarrow$ W
  - W $\rightarrow$ W
  - W $\rightarrow$ R
  - S $\rightarrow$ W
  - S $\rightarrow$ R
  - W $\rightarrow$ S
  - R $\rightarrow$ S
- preserved = operations appear to be completed in order to the program
- relaxed = a later operation can complete first
- program correctness requires that the order be preserved when operations are to the same address
- the consistency model determines which orders must be preserved

Sequential consistency: all 8 orderings are preserved

Memory Consistency Models

Nonconflicting reads can pass writes within a critical section

- **processor consistency**
- **total store order** (TSO)
- HW implementation
    - write buffer that checks if read/write address match

Nonconflicting writes can pass writes within a critical section

- **partial store order** (PSO)
- HW implementation
    - pipelining
    - write back memory update

Memory Consistency Models

Nonconflicting accesses of any type can pass reads within a critical section

- only synchronization operations are sequentially consistent
- **weak ordering**
- HW implementation
    - nonblocking processor
    - out-of-order execution
    - critical sections

Synchronization operations are relaxed

- acquires can pass reads & writes
- reads & writes can pass releases
- acquires can pass releases
- only critical sections are enforced
    - acquires must preceed use of shared data
    - release must follow updates to shared data
    - release must preceed the next acquire
- **release consistency**
- HW implementation
    - directory counts invalidations to make sure all writes complete

Memory Consistency Models

Implementation

- **Alpha**
  - release consistency
  - memory barrier instruction (the fence)
  - placed at beginning & end of critical sections
- **MIPS-IV**
  - release consistency
- **V9**
  - `membar`: ld/st options for issue/completion
  - TSO, PSO, one weaker
- **Pentium**
  - serializing instructions: `cpuid, wbinvd`
  - processor consistency