A Primer on Memory Consistency and Cache Coherence (Chapter 7-8)

by Yunqi Zhang
04/22/2013

1. Snooping Coherence Protocols
    a. basic idea: broadcast requests to ***all*** controllers in ordered network - ensures that every controllers observe the same series of coherence requests in same order
    b. baseline system state
        i. stable state: M(odified), S(hared), I(nvalid)
        ii. transient state: coherence request has been sent but hasn't received response
    c. atomic request: ensures when a cache controller seeks to upgrade permissions to a block, it can issue a request without worrying another core's request might be ordered ahead
    d. atomic transaction: no subsequent requests for a block will occur until after the current transaction completes

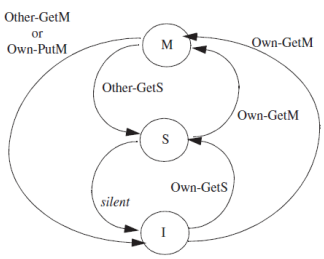

FIGURE 7.1: MSI: Transitions between stable states at cache controller.

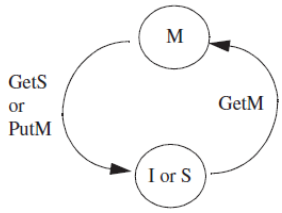

FIGURE 7.2: MSI: Transitions between stable states at memory controller.

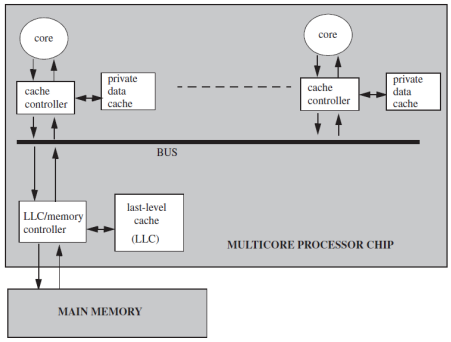

FIGURE 7.3: Simple snooping system mode.

2. Optimization
    a. adding the **exclusive state**: optimize for the case that a core first read a block and then subsequently writes it
        i. MSI: 1) GetS to read, 2) GetM to write
        ii. MESI: 1) GetS to read / GetE if no others have access, [2) GetM to write]
    b. 2 possible solutions to request for Exclusive State
        i. implement a wired-OR signal through the bus, all controllers assert the signal
        ii. maintain extra state at LLC
    c. adding the **owned state**: optimize for the case that a cache has a block in state M or E and receives a GetS from another core
        i. MSI: change from M/E to S and send data to both requestor & memory controller
        ii. MOSI: eliminates 1) sending data to update LLC/MEM, 2) unnecessary write to LLC [3) allows subsequent requests to be satisfied by cache instead of LLC]
    d. **non-atomic bus**
        i. pipelined bus: in-order, blocked before receive the previous one's response
        ii. split-transaction bus: out-of-order, low-latency, need to match responses with

requests

e. possible issues of split-transaction bus
    i. many more possible transitions at both cache and memory controllers
    ii. stalling raises the potential of deadlock

3. Optimizations to the bus interconnection network
    a. motivation: no need to order coherence responses nor broadcast them
    b. separate non-bus interconnection network for coherence responses
        i. implementability: hard to implement high-speed shared-wire busses
        ii. throughput: multiple responses in-flight at a time
        iii. latency: allows responses to be sent immediately without arbitration
4. Directory Coherence Protocols
    a. key idea: establish directory that maintains a global view of coherence state of each block
        i. unicast coherence requests, send requests to directory
        ii. transaction order determined by the order they are serialized at the directory
        iii. conflicting requests are processed by all nodes in per-block order
    b. deadlock avoidance
        i. motivation: the reception of a message can cause to send another message
        ii. use physically or logically separate networks to send different classes of message
            1. e.g. request messages, forward messages, response messages
    c. directory state representation
        i. significant storage space to maintain the states of the full set of the caches
        ii. coarse directory
            1. maintain a list of sharers that is superset of the actual one
            2. cost extra interconnection network bandwidth for unnecessary messages
        iii. limited pointer directory
            1. limited number of pointers, need to handle the unexpected case by
                a. broadcast the message to all controllers
                b. invalidate one of the current sharers
                c. trap a software handler with performance cost
    d. directory cache: provides faster access to a subset of the complete directory state
        i. most straightforward implementation: directory cache backed by DRAM
        ii. inclusive directory caches: holds entries for superset of all blocks cached on chip
        iii. null directory caches: having no directory cache at all
    e. performance and scalability optimizations
        i. distributed directory: directory for a given block is fixed in one place, but different blocks can have different directories
        ii. non-stalling directory: need large number of transient states to avoid stalling
        iii. interconnection networks without P2P ordering: need handshake messages to eliminate races
        iv. silent evictions of blocks in state S: reduces interconnection network bandwidth