Cache Coherence

Ryan Lee

6.823 Fall 2023

Adapted from prior course offerings

Goals of caches

» Small memories that provide quick access to recently accessed data.

» Transparently managed by hardware (and OS)
  - Program output should appear as if the caches did not exist and applications directly accessed main memory.
  - In contrast with scratchpads (explicitly managed)

Goals of shared memory

» Multiple concurrently executing threads can read and write data in a single address space.

» Transparently managed by hardware (and OS)
  - Program output should appear as if the caches did not exist and applications directly accessed single memory.
  - In contrast with message passing (explicitly manage shared data)

Caches in parallel systems

» Caches give quick access to data:
  - Small **private caches** may hold copies of data.

» Transparent management: How to ensure cache accesses don't act on stale data?
  - No shared writeable address space: Pure message passing, or
  - Cache coherence

Cache coherence

time

Processor X:	Processor Y:
Ld 0xA → 0	
Ld 0xA → 0	
	St 42 → 0xA
Ld 0xA → 0	
…	
Ld 0xA → 42	
Ld 0xA → 42	

Q. Do you think this system is coherent?

Cache Coherence

» Two Rules:

1. Write propagation: Writes **eventually** become visible to other processors
2. Write serialization: All processors observe writes to one location appear to happen in a consistent order

» Strategies for propagation:

- A write **invalidates** copies in other private caches
- A write **updates** copies in other private caches
- Tradeoffs?

Serialization strategies

» **Snoopy** coherence protocol
On a miss, private caches broadcast their actions through a bus-like interconnect, other caches observe ("snoop") and perform updates or invalidations.

» **Directory-based** coherence protocol
On a miss, private caches send unicast message to the directory, which serializes requests and sends unicast messages to other caches to perform updates or invalidations.

Tradeoffs?

Do write-through caches need coherence?

» **Yes**.

- Writes must propagate: update or invalidate copies in other private caches.
- Write serialization is trivial (where is the serialization point?)

» A protocol with two stable states is sufficient:

- Invalid
- Shared

» Do you need transient cache states?

- Yes!

Write-back caches: MSI

» Three stable states per cache-line
- Invalid (I): Cache does not have a copy
- Shared (S): Cache has read-only copy; clean
- Modified (M): Cache has only copy; writable; (potentially) dirty

» Processor-initiated actions:
- Read: needs to upgrade permission to S
- Write: needs to upgrade permission to M
- Evict: relinquish permissions (caused by access to a different cache line)

Optimizations

» Problem: Writeback to memory upon M->S downgrade
  - Sometimes wastes bandwidth e.g. producer-consumer scenarios
  - S implicitly assumes line is clean, allowing silent evictions.

» Solution: Add Owner (O) state
  - O: Multiple copies, read-only, and dirty. Also responsible for writing back the data
  - Core entres O upon a downgrade.

Lab Task: MSI Coherence Protocol

» Implement with Murphi description language
  - Rules: Define transitions between states
  - Invariants and asserts: Capture protocol correctness

» Murphi verifier
  - Explores reachable states until it finds:
    - A violation of an invariant or assertion, or
    - A state with no possible transitions (deadlock), or
    - It has explored all reachable states and found no errors.
  - Exploits symmetry to reduce redundant states

Races

- Occur when there are multiple messages/requests in flight concerning a single cache line.
- Try to minimize the opportunity for races by waiting for previous messages before sending new ones.
- Multiple processors may concurrently initiate conflicting requests.
- If network may deliver messages out of order, the protocol must handle this. For example:
  - The directory has two messages in flight to one private cache.
  - One processor/cache has two messages in flight to the directory.
- 3-hop protocol may require you to add more handling for additional races.

Tips

» Feel free to add to or rename states and messages.

» Get a 4-hop protocol working first, before attempting 3-hop.

» Get your protocol working with ProcCount set to 2 before handling the 3-processor case.

» Write more of assertions and/or invariants.

  - Add assertions/invariants about your transient states.