

Cache Coherence

Victor Ying

6.823 Spring 2020

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.

Goals of shared memory

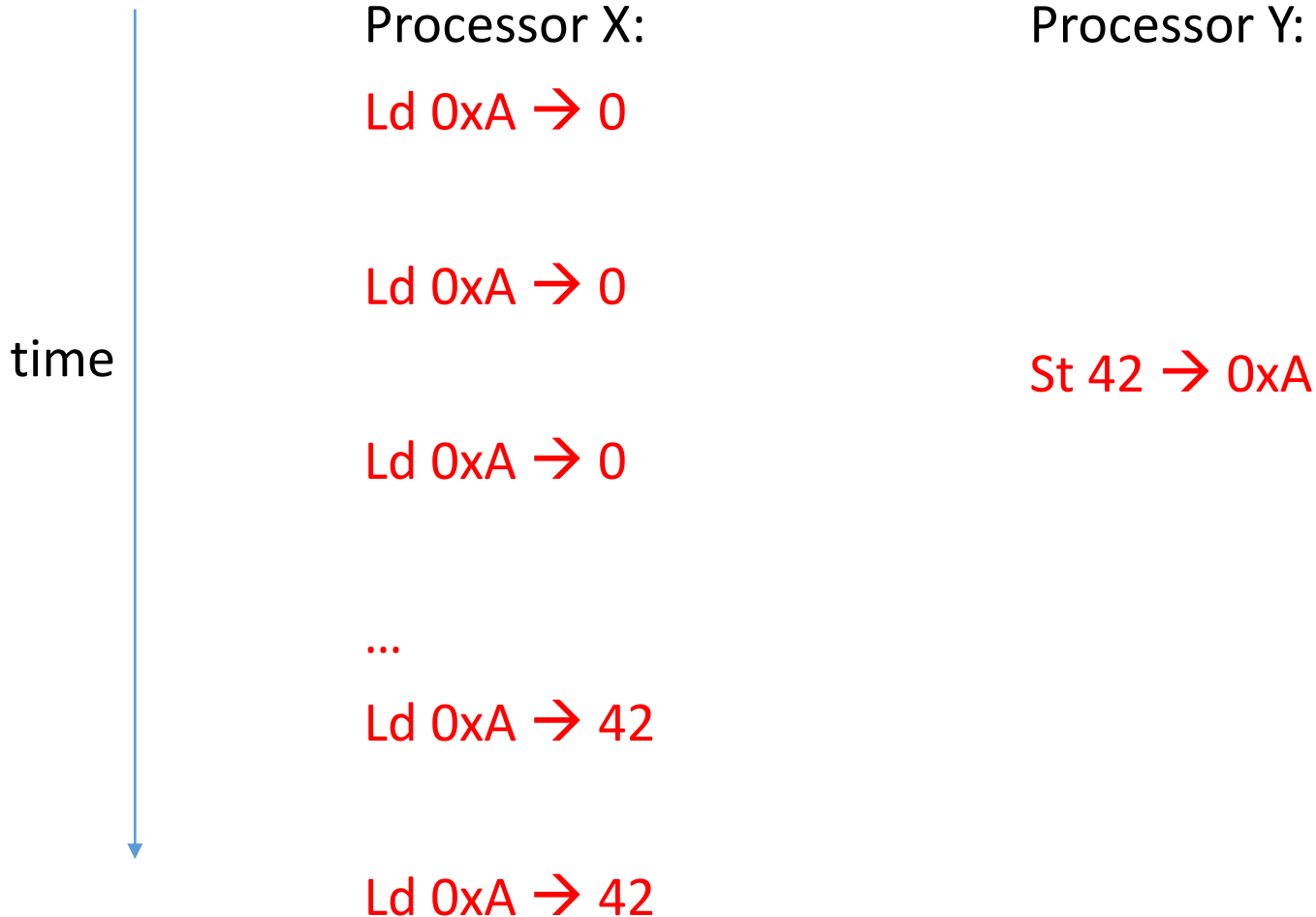
- » Multiple concurrently executing threads can read and write data in a single address space.
- » Transparently managed by hardware (and OS)
 - Program output should appear as if the caches did not exist and applications directly accessed single memory.

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



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.

» A protocol with two stable states is sufficient:

- Invalid
- Shared

» Do you need transient cache states?

- Implemented with MSHR

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)

MSI directory states

- » Uncached (Un): No cache has a valid copy
- » Shared (Sh): One or more caches in S state. Must track sharers.
- » Exclusive (Ex): One of the caches in M state. Must track owner.

- » Does the directory need transient states?
 - Yes on downgrades/invalidations, to guarantee serialization

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.
 - L15-24 shows one case
- » 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.