An Aside

Course thus far:
We have focused on *parallelism*: how to divide up work into parallel tasks so that they can be done faster.

Rest of the term:
We will focused on *concurrency*: how to correctly and efficiently manage access to shared resources.
Memory Coherence
and Memory Consistency

Acknowledgement:
- slides adopted from Kayvon Fatahalian at CMU
- materials taken from "A Primer on Memory Consistency and Cache Coherence" by Danial J. Sorin, Mark D. Hill, and David A. Wood
The Shared Memory Abstraction

Programmer’s intuition: memory is coherent.
The Problem of Coherence

load x

P

P

P

P

$\$

$\$

$\$

$\$

x: 0

Memory
The Problem of Coherence

load x

P
x = 0

load x

P
x = 0

x: 0

Memory
The Problem of Coherence

Coherence problems exist because the abstraction of shared address space is implemented by both a global storage (main memory) and per-processor local storage (private caches).
Cache Hierarchy of Intel Core i7 CPU

- **L1 Data Cache**: 32 KB, 8-way set associative, write back, 2 x 16B loads + 1 x 16B store per clock, 4-6 cycle latency, up to 10 outstanding misses.
- **L2 Cache**: 256 KB, 8-way set associative, write back, 32B / clock, 12 cycle latency, up to 16 outstanding misses.
- **Shared L3 Cache**: 8 MB, inclusive, 16-way set associative, 32B / clock per bank, 26-31 cycle latency.

Ring Interconnect
Definition of Coherence

Dividing a memory location’s lifetime into epochs, where

1. Every epoch has a either a single writer or multiple readers
   (write serialization: ex: P1 writes 1 to x and P2 write 2 to x, the values are observed in the same order).

2. The value of the memory location propagate from the end of one epoch to the beginning of the next epoch
   (value propagation: the new value eventually gets to other processors.).

A cache coherence protocol maintains these two invariants.
(The granularity of coherence is a cache line size.)
Memory Coherence vs Memory Consistency

• Coherence only defines the behavior of reads and writes to the same memory location.
  - It does not define a global ordering among reads and writes to different memory locations from two different processors.
  - You cannot put the two memory operations on a single timeline and have both processor’s observations agree with the timeline).

• Memory consistency (later) defines the behavior of reads and writes to different memory locations.
Implementing Coherence

We will assume:

• private caches
• write-back caches (a dirty line gets written back to memory upon eviction)
• upon a memory access (load or store), the memory gets brought into the cache.

We will focus on hardware cache coherence protocol:

• snooping
• directory
Implementing Coherence

Cache line:

<table>
<thead>
<tr>
<th>Line state</th>
<th>Tag</th>
<th>Data (64 bytes on Intel Core i7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dirty bit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cache coherence protocol uses the state bits to maintain a state machine per cache line.

Possible cache line states:

- **Modified (M):** the block is valid, exclusive, owned by this cache, and potentially dirty.
- **Shared (S):** the block is valid and read-only (not exclusive).
- **Invalid (I):** the block is invalid.
- **Exclusive (E):** the block is valid, exclusive, and clean (an optimization; not strictly necessary).

(There is also "Owned" and "Forward" states, but we will ignore those.)
Snooping Protocol
System Model for Snooping Cache

Broadcast request on interconnect, and all the controllers collectively do the right thing.
System Model for Snooping Cache

- P
- cache controller
- L1 L2 cache
- L1 L2 cache
- Interconnect
- LLC/memory controller
- last-level cache (LLC)
- Main memory
- Multicore processor chip
- load / store

- Multicore processor chip
- Main memory
- Interconnect
System Model for Snooping Cache

- Examine cache line state and send out request

Diagram showing a system model with cache controllers, L1 L2 caches, LLC/memory controller, last-level cache (LLC), and main memory.
System Model for Snooping Cache

Maintain line state from the cache's point of view: M means some cache owns it; otherwise not.
Basic MSI Invalidation Protocol

• Key tasks of protocol
  – obtain exclusive access for a write
  – locating most recent copy of data on a cache miss

• Cache line state: M, S, I

• Processor events:
  – PrRd (load)
  – PrWr (store)

• Interconnect events:
  – BusRd: obtain a copy with no intent to modify
  – BusRdX: obtain a copy with intent to modify
  – BusWB: write the line back to memory
MSI State Transition Diagram

A/B: if action A is observed by cache controller, action B is taken.

- : initiated by other processors
- : initiated by this processor

flush : write dirty line back to memory
Cache Controller
MSI State Transition Diagram

A/B: if action A is observed by cache controller, action B is taken.

- ---: initiated by other processors
- ----: initiated by this processor

flush: write dirty line back to memory
It also sends the data if it's cache line state is in S or I.
Does MSI Satisfy Coherence?

• Write serialization:
  – two writes from different processors are serialized by the order their requests arrive to interconnect (crucial: we assume atomic transaction).
  – the protocol maintains single-writer or multiple readers at a time.

• Value propagation:
  – achieved via a combination of invalidation on BusRdX and flush from M-state on subsequent BusRd / BusRdX from another processor.
MESI Invalidation Protocol

• MSI requires two bus transactions for the common case of reading data then subsequently writing to it:
  – BusRd to move from I to S
  – BusRdX to move from S to M

• Adding an E state:
  – upgrading from E to M does not require any bus transaction
  – cache line is clean (not modified yet), but this cache has the only copy:
    • LLC explicitly distinguish between I and S (conservative or we'd require additional transaction upon S line eviction).
Lower-Level Choices

• Who should supply data on a cache miss when line is in the E or S state of another cache?
  – Can get it either from LLC / Memory control or from another cache controller.
  – If from another cache controller, which cache controller should provide it?
    (why there is "MESIF" and "MOEIS")

• Cache-to-cache transfers add complexity, but commonly used to reduce latency of access and memory bandwidth.
False sharing

• Two processors write to different memory locations that map to the same cache line.
• Each processor gets it in M state whenever it needs to write, so the cache line "ping-pong" between the caches of the writing processors.
• An artifact of the fact that we maintain coherence with cache line granularity.
• Your program performance will suck.