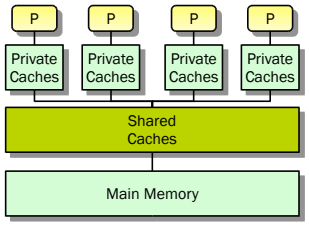
Chapter 5

Thread-Level Parallelism

From ILP to TLP

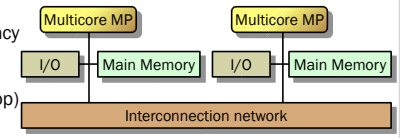
- ILP became inefficient in terms of
  - Power consumption
  - Silicon cost
  - Workloads did not justify it
    - Independent computations on large data sets dominate
- Thread-Level parallelism is a form of MIMD
  - Uses MIMD model
  - Have multiple program counters
  - Targeted for tightly-coupled shared-memory multiprocessors
- For N processors, need N threads
- Amount of computation assigned to each thread is grain size
  - Threads can be used for data-level parallelism, but the overheads may outweigh the benefit

Multiprocessor Types

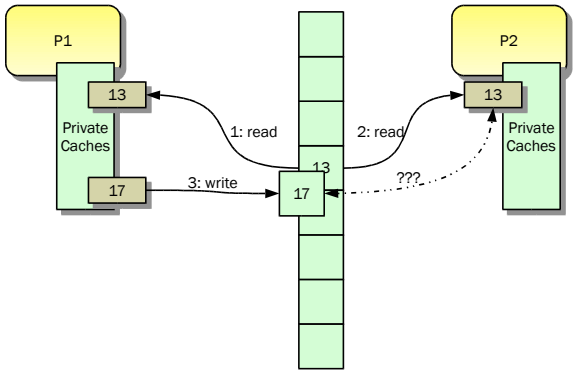


Centralized shared-memory
- Symmetric Multiprocessors (SMP)
- Small number of cores
- Shared single memory
- Uniform main memory latency

Distributed shared-memory
- Memory distributed among processors
- Non-uniform memory access/ latency (NUMA)
- Processors connected via direct (switched) and non- direct (multi-hop) interconnection networks



Multiprocessor Cache Coherence



Memory System is Coherent If…

- P writes X, P reads X, no intervening writes
  - X returns value written by P
- Q writes X, <sufficient time period>, P reads
  - X returns value written by Q
- Writes are serialized
  - Written values seen in the same order by all processors
- Coherency defines what values can be returned by a read
- Consistency defines when a written value is returned by a read
  - If processor P writes to location A and than to B then any processor Q that sees the new value in B must also see the new value in A

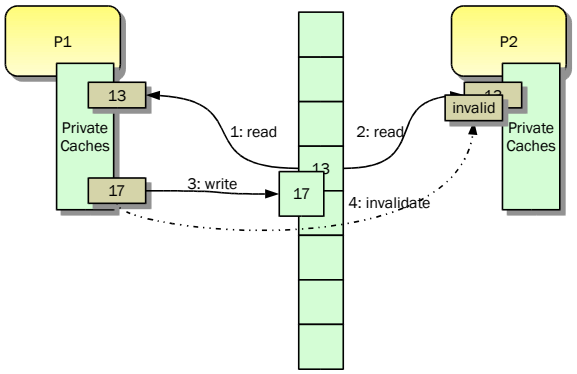
Enforcing Cache Coherence

- Coherent caches provide:
  - Migration: movement of data
  - Replication: multiple copies of data
- Cache coherence protocols
  - Directory based
    - Sharing status of each block kept in one location
    - Single directory for small core counts, or
    - Distributed directories
  - Snooping
    - Each core tracks sharing status of each block
    - Requires broadcast medium, e.g. data bus
    - May be combined with cache directories for large core counts

Snooping Cache Protocols

- Write invalidate protocol
  - Most common protocol
  - Upon write, make sure there is only one valid copy
  - Write serialization happens through exclusive access to data bus
- Write update protocol (write broadcast)
  - Upon write, make sure all copies are the same
  - Requires more bandwidth

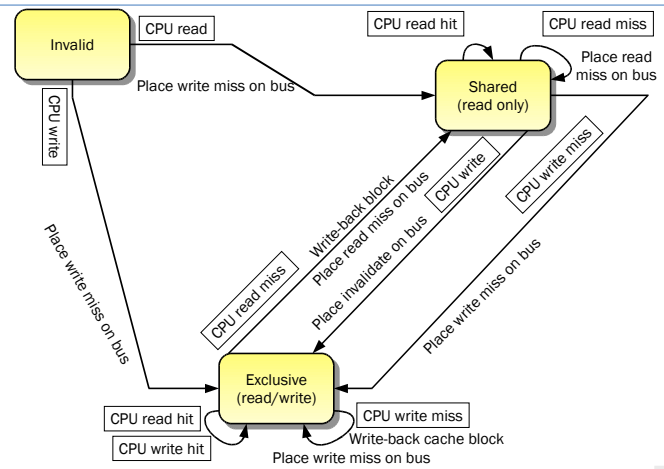
Write Invalidate Cache Coherence Protocol



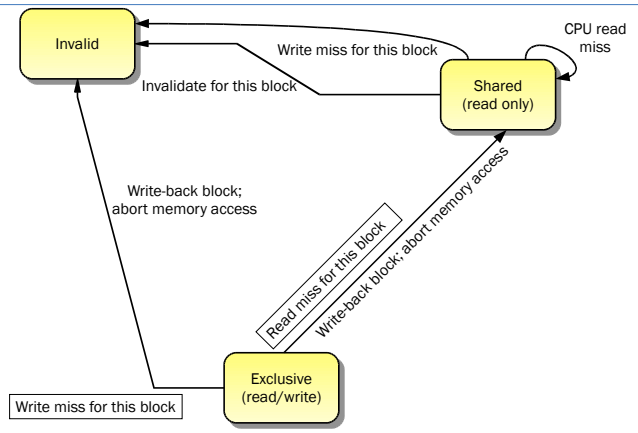
Snooping Protocol Implementation Details

- Where is the most recent value of data?
  - Write-through cache: in shared cache or memory
  - Write-back cache: in other core's private cache or in shared cache/memory
- Getting the most recent value of data
  - If in shared-cache/memory: request as usual
  - If in private cache:
    - Ask for data from other core and wait
    - Invalidate data in other core's cache
- What to do with the data buffers
  - Need to be treated as another cache

State Transition for Write Invalidate: CPU Requests



State Transition for Write Invalidate: Bus Requests



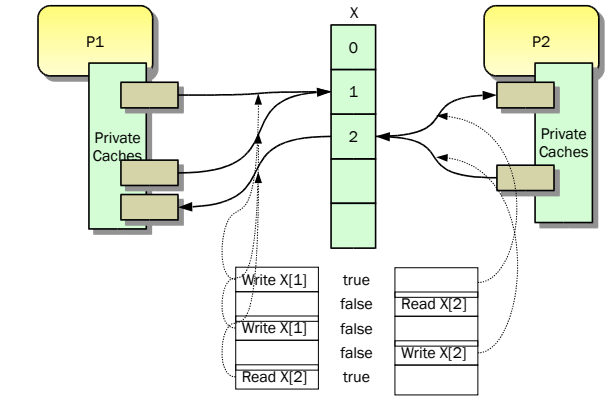
Snooping Protocol Extensions

- Basic protocol (Modified-Shared-Invalid = MSI) has problems
  - State transitions are not atomic
    - They involve many steps
      - detect change/request of data
      - acquire bus
      - send request
      - receive response
  - Multi-processor (multi-socket) systems need intra- and inter-chip solution
    - Intel: Quickpath Interconnect
    - AMD: Hypertransport

Extensions to Simple Coherence Protocols

- MESI = Modified-Exclusive-Shared-Invalid
  - Coherent bit is added for each cache block
  - Dirty bit indicates that the block was modified
- MESIF = Modified-Exclusive-Shared-Invalid-Forward (Intel i7)
  - Forward state: if a core has a block in this state it will respond to misses
    - Prevents all cores from responding to a single miss
- MOESI (AMD)
  - Owned state indicates that the cache "owns" the block and the memory is out-of-date
  - Transition M→O prevents a (costly) write to memory
  - Note: blocks can be shared in M state

True and False Sharing



True sharing:
- Write to shared block
- Read invalidated block

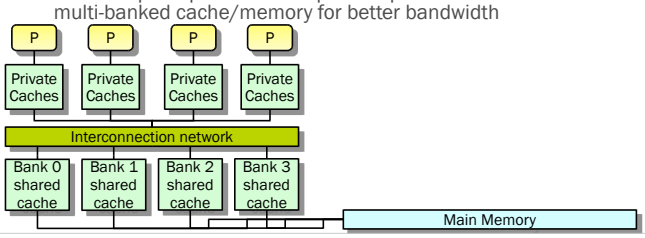
False sharing:
- Read unmodified word in invalidated block

Limitations of SMP and Snooping Protocols

- Practical scalability limit: 8 cores
- Scalability bottleneck: snooping bandwidth

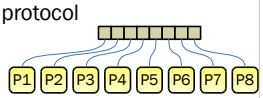
Larger cache cannot alleviate snooping broadcasts: faster processor means more traffic.

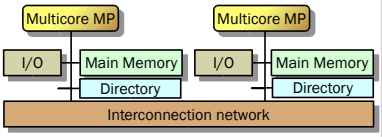
  - Solution: duplicate address tags to keep private cache separate
    - One set of tags for internal cache operation
    - One set of tags for comparisons when misses happen on snooping bus
  - Solution: add a directory to the outermost cache
    - The directory maps the blocks to the processors that own it
  - Solution: pair up cross-bar or point-to-point interconnects with multi-banked cache/memory for better bandwidth



Directory-Based Coherence

- Snooping broadcast are a major drain of cache bus bandwidth
- Distribution of resources localizes the traffic
  - But snooping broadcasts must be removed
- Alternative to snooping is the directory protocol
- Directory keeps
  - The state of every cache block
  - Which caches have a copy of the block
  - It could be as simple as an shared L3 cache that is inclusive
    - Each L3 block as a bit vector of length=number of cores
- Single directory scheme is not scalable
  - Directory must be distributed





Snooping Protocol Extensions: AMD Opteron

- Snooping inside each multicore chip
- Separate memory connected to each multicore chip
  - NUMA organization
- Coherence implemented with point-to-point links with up to three other chips
  - The links not part of the bus
  - Broadcast on each link the request for a cache block (like snooping)
  - Acknowledgement completes memory operations
    - Similar to directory-based protocols
    - Used for serialization of operations

Directory Protocol Basics

- Storage size
  - Number of cached memory blocks * number of nodes
    - Node is a single core, a single multicore socket, or a collection of multicore chips with internal coherence (usually based on snooping)
- For each block, directory maintains a state. For example:
  - Shared
    - One or more nodes have the block cached, value in memory is up-to-date
    - Set of node IDs or a bit vector
  - Uncached
    - No node has a copy of the cache block
    - It's been evicted or it's somewhere in lower (below directory) level caches
  - Modified
    - Exactly one node has a copy of the cache block, value in memory is out-of-date
    - Owner node ID
- Directory maintains block states and sends invalidation messages

Directory Protocol Terms

- Protocol has two sides
  - Local (originator)
  - Remote
- Local node is where the requests originates
- Home node is where memory location (address) and is directory entry reside
- Invalid state
  - The state of a cache block from the perspective of that block
- Uncached state
  - The state of a cache block from the perspective of the directory

Directory Protocol Messages

Message type	Source	Destination	Message contents	Function of this message
Read miss	Local cache	Home directory	P, A	Node P has a read miss at addr. A; request data and make P share A
Write miss	Local cache	Home directory	P, A	Node P has a write miss at addr. A; request data and make P exclusive owner
Invalidate	Local cache	Home directory	A	Request to send "invalidate" to all remote caches that cache A
Invalidate	Home directory	Remote cache	A	Invalidate a shared copy of data at A
Fetch	Home directory	Remote cache	A	Fetch the block at A and send it to its home directory; Change the state of A in remote cache to "shared"
Fetch/Invalidate	Home directory	Remote cache	A	Fetch the block at A and send it to its home directory; Invalidate the block in the cache
Data value reply	Home directory	Local cache	D	Return a data value from the home memory
Data write-back	Remote cache	Home directory	A, D	Write-back a data value for address A

Directory Protocol Details: Uncached and Shared

- For uncached block:
  - Read miss
    - Requesting node is sent the requested data and is made the only sharing node, block is now shared
  - Write miss
    - The requesting node is sent the requested data and becomes the sharing node, block is now exclusive
- For shared block:
  - Read miss
    - The requesting node is sent the requested data from memory, node is added to sharing set
  - Write miss
    - The requesting node is sent the value, all nodes in the sharing set are sent invalidate messages, sharing set only contains requesting node, block is now exclusive

Directory Protocol Details: Exclusive

- For exclusive block:
  - Read miss
    - The owner is sent a data fetch message, block becomes shared, owner sends data to the directory, data written back to memory, sharers set contains old owner and requestor
  - Data write back
    - Block becomes uncached, sharer set is empty
  - Write miss
    - Message is sent to old owner to invalidate and send the value to the directory, requestor becomes new owner, block remains exclusive

Hardware Synchronization Overview

- There is always a need for synchronizing threads
  - Start, stop, increment a shared counter, …
- Software-based solutions are slower, reserved for software sync.
- Basic synchronization concepts: atomicity and transaction
  - Concepts also used in databases
- Elementary synchronization primitives
  - Atomic exchange
    - Swaps a register with a memory location
  - Test-and-set
    - Set a value if a condition is met
  - Fetch-and-increment
    - Returns a value in memory and increments after the read
- Requirement: read and write as a single, uninterruptable instruction
  - Coherence protocol must guarantee atomicity and lack of deadlock or livelock

Hardware Synchronization: Implementation

- Special instruction with atomic semantics
  - Requires complications to coherence protocol for caches, directories,…
- Special instruction pair: "attempt" and "see if succeeded"
- Example:
  - Load linked (or load locked)
    - Tries to load the content of a memory location A
  - Store conditional
    - It might succeed if there was no intervening change to the location A
    - It fails if a write to A occurred or context switch occurred
  - Sample implementation of atomic exchange between R4 and addr(R1)

```
try:    mov       R3, R4     ; moves between registers are atomic
        LoadLink  R2, 0(R1)  ; initiate a load
        StoreCond R3, 0(R1)  ; return 0 on failure
        beqz      R3, try    ; keep trying if failure
        mov       R4, R2     ; R2 got the value from LoadLink
```

    - Is the memory still coherent? Is deadlock possible? Livelock?

Memory Consistency Introduction

- How consistent memory view must be?
- What is the observable order of writes from different processors
  - Observation = read from memory
  - Consistency defines properties between reads and writes
- Strongest form of consistency: sequential consistency

Memory Consistency: Example

- Processor 1
  A = 0
  …
  A = 1
  if (B == 0) …
- Processor 2
  B = 0
  …
  B = 1
  if (A == 0) …

- What does A=0 do?
  - If A is not in a cached block: send request to get it
  - Send invalidate if A is in shared block in some processor
  - Wait
    - For block with A
    - For invalidate to complete

Memory Consistency: Sample Interleaved Execution

- Processor 1
  A = 0
  …
  A = 1
  if (B == 0) …
- Processor 2
  B = 0
  …
  B = 1
  if (A == 0) …

A=0
A=1
if (B==0) …

B=0
B=1
if (A==0)…

A=0
A=1
if (B==0) …

B=0
B=1
if (A==0)…

B=0
B=1
if (A==0)…

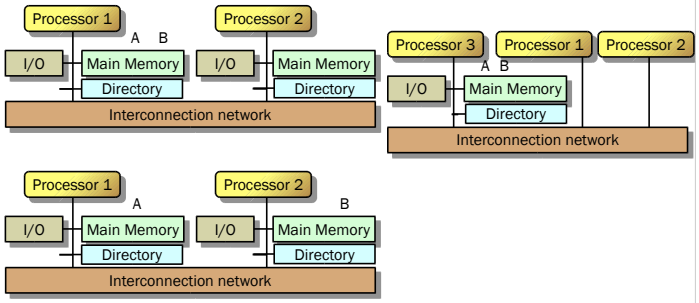
A=0
A=1
if (B==0) …

B=0
B=1
if (A==0)…

How many possible interleaved sequences?

Memory Consistency: Coherence Protocol View

- Processor 1
  A = 0
  …
  A = 1
  if (B == 0) …
- Processor 2
  B = 0
  …
  B = 1
  if (A == 0) …



Strongest Form of Memory Consistency

- Sequential consistency requires…
  - accesses from each processor are kept in order
  - accesses among different processors are arbitrarily interleaved
- Sample implementation
  - Delay each memory access until all invalidation messages complete, or
  - Delay the next memory access until the previous one completes
  - Either scheme affects performance negatively
    - "delay" means more memory-related stalls
- To improve performance
  - Make the coherence protocol faster and more effective by hiding latency etc.
  - Relax the consistency model
    - More stringent consistency model may be implemented in software and only as required

Relaxed Consistency Models

- Rule X → Y means that …
  - Operation X must complete before operation Y is done
- Sequential consistency requires all orderings to be maintained:
  - R → W, R → R, W → R, W → W
- Relaxed W → R
  - Is called: Total store ordering or processor consistency
- Relaxed W → W
  - Is called: Partial store order
- Relaxed R → W and R → R
  - Is called: Weak ordering, PowerPC consistency model, and release consistency
- Relaxing any of the orderings is done to increases performance