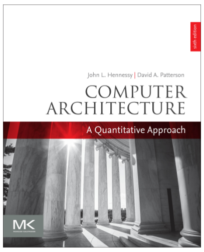
Computer Architecture
A Quantitative Approach, Sixth Edition



Chapter 5

Thread-Level Parallelism

 1

Introduction

Introduction

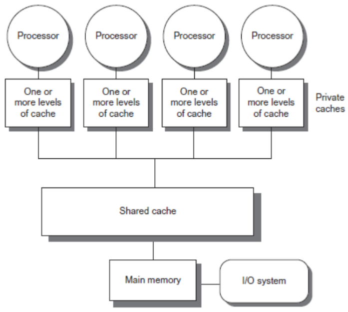
- Thread-Level parallelism
  - Have multiple program counters
  - Uses MIMD model
  - Targeted for tightly-coupled shared-memory multiprocessors
- For *n* processors, need *n* threads
- Amount of computation assigned to each thread = grain size
  - Threads can be used for data-level parallelism, but the overheads may outweigh the benefit

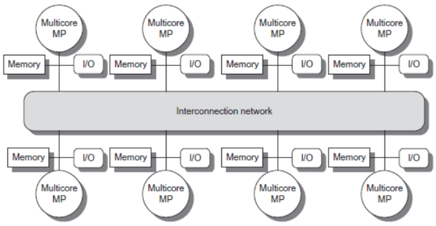
 2

Types

Introduction

- Symmetric multiprocessors (SMP)
  - Small number of cores
  - Share single memory with uniform memory latency
- Distributed shared memory (DSM)
  - Memory distributed among processors
  - Non-uniform memory access/latency (NUMA)
  - Processors connected via direct (switched) and non-direct (multi-hop) interconnection networks





 3

Cache Coherence

Centralized Shared-Memory Architectures

- Processors may see different values through their caches:

Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

 4

Cache Coherence

- Coherence
  - All reads by any processor must return the most recently written value
  - Writes to the same location by any two processors are seen in the same order by all processors
- Consistency
  - When a written value will be returned by a read
  - If a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A

Centralized Shared-Memory Architectures

 5

Enforcing 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
  - Snooping
    - Each core tracks sharing status of each block

Centralized Shared-Memory Architectures

 6

Snoopy Coherence Protocols

- Write invalidate
  - On write, invalidate all other copies
  - Use bus itself to serialize
    - Write cannot complete until bus access is obtained

Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
				0
Processor A reads X	Cache miss for X	0		0
Processor B reads X	Cache miss for X	0	0	0
Processor A writes a 1 to X	Invalidation for X	1		0
Processor B reads X	Cache miss for X	1	1	1

- Write update
  - On write, update all copies

 7

Snoopy Coherence Protocols

- Locating an item when a read miss occurs
  - In write-back cache, the updated value must be sent to the requesting processor
- Cache lines marked as shared or exclusive/modified
  - Only writes to shared lines need an invalidate broadcast
    - After this, the line is marked as exclusive

 8

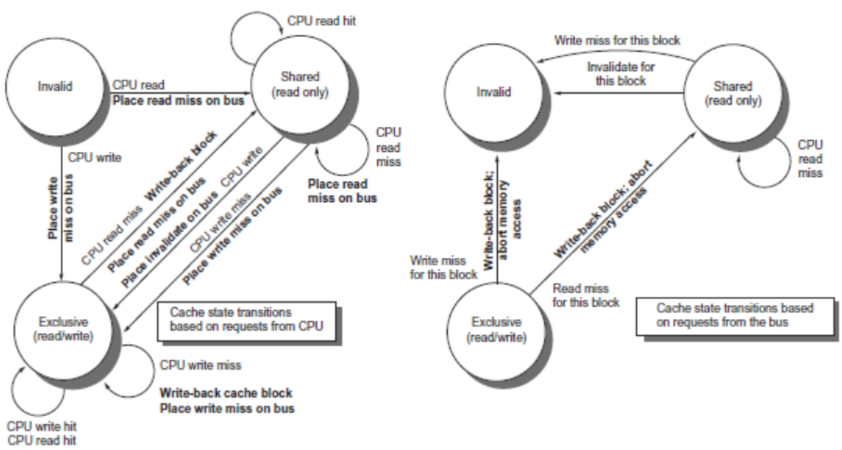
Snoopy Coherence Protocols

Request	Source	State of addressed cache block	Type of cache action	Function and explanation
Read hit	Processor	Shared or modified	Normal hit	Read data in local cache.
Read miss	Processor	Invalid	Normal miss	Place read miss on bus.
Read miss	Processor	Shared	Replacement	Address conflict miss: place read miss on bus.
Read miss	Processor	Modified	Replacement	Address conflict miss: write-back block; then place read miss on bus.
Write hit	Processor	Modified	Normal hit	Write data in local cache.
Write hit	Processor	Shared	Coherence	Place invalidate on bus. These operations are often called upgrade or *ownership* misses, because they do not fetch the data but only change the state.
Write miss	Processor	Invalid	Normal miss	Place write miss on bus.
Write miss	Processor	Shared	Replacement	Address conflict miss: place write miss on bus.
Write miss	Processor	Modified	Replacement	Address conflict miss: write-back block; then place write miss on bus.
Read miss	Bus	Shared	No action	Allow shared cache or memory to service read miss.
Read miss	Bus	Modified	Coherence	Attempt to read shared data: place cache block on bus, write-back block, and change state to shared.
Invalidate	Bus	Shared	Coherence	Attempt to write shared block; invalidate the block.
Write miss	Bus	Shared	Coherence	Attempt to write shared block; invalidate the cache block.
Write miss	Bus	Modified	Coherence	Attempt to write block that is exclusive elsewhere; write-back the cache block and make its state invalid in the local cache.

9

Centralized Shared-Memory Architectures

Snoopy Coherence Protocols



10

Centralized Shared-Memory Architectures

Snoopy Coherence Protocols

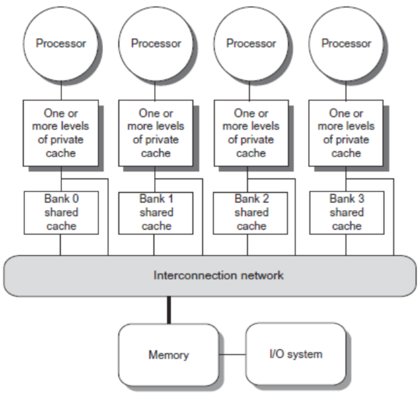
- Complications for the basic MSI protocol:
  - Operations are not atomic
    - E.g. detect miss, acquire bus, receive a response
    - Creates possibility of deadlock and races
    - One solution: processor that sends invalidate can hold bus until other processors receive the invalidate
- Extensions:
  - Add exclusive state to indicate clean block in only one cache (MESI protocol)
    - Prevents needing to write invalidate on a write
  - Owned state

Centralized Shared-Memory Architectures

 11

Coherence Protocols: Extensions

- Shared memory bus and snooping bandwidth is bottleneck for scaling symmetric multiprocessors
  - Duplicating tags
  - Place directory in outermost cache
  - Use crossbars or point-to-point networks with banked memory



Centralized Shared-Memory Architectures

 12

Coherence Protocols

- Every multicore with >8 processors uses an interconnect other than bus
  - Makes it difficult to serialize events
  - Write and upgrade misses are not atomic
  - How can the processor know when all invalidates are complete?
  - How can we resolve races when two processors write at the same time?
  - Solution: associate each block with a single bus

Centralized Shared-Memory Architectures

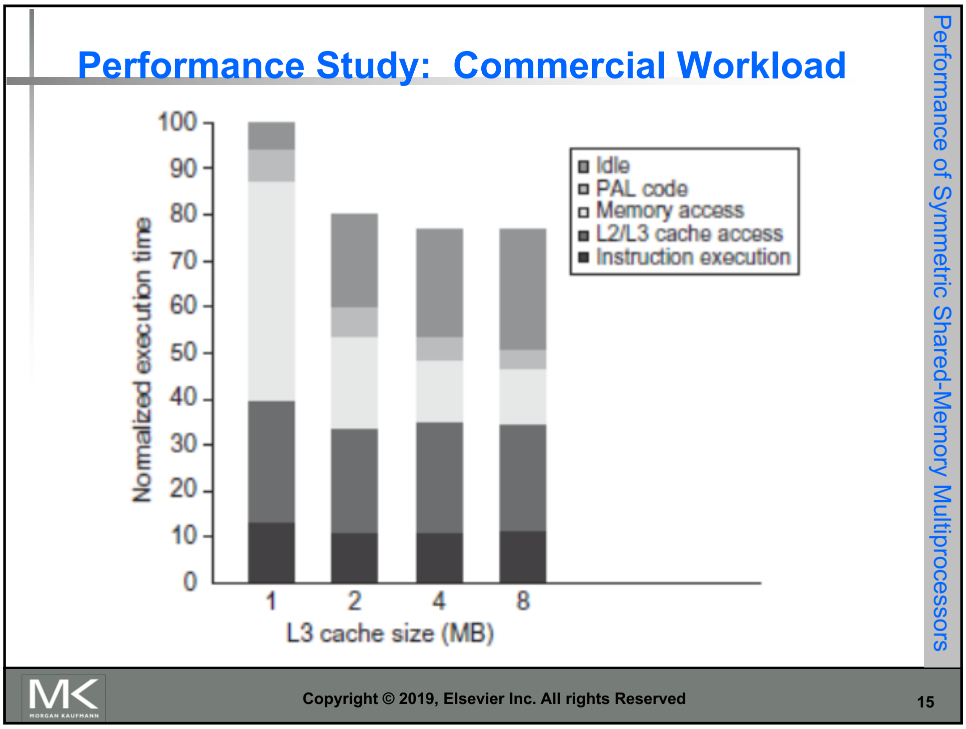
 13

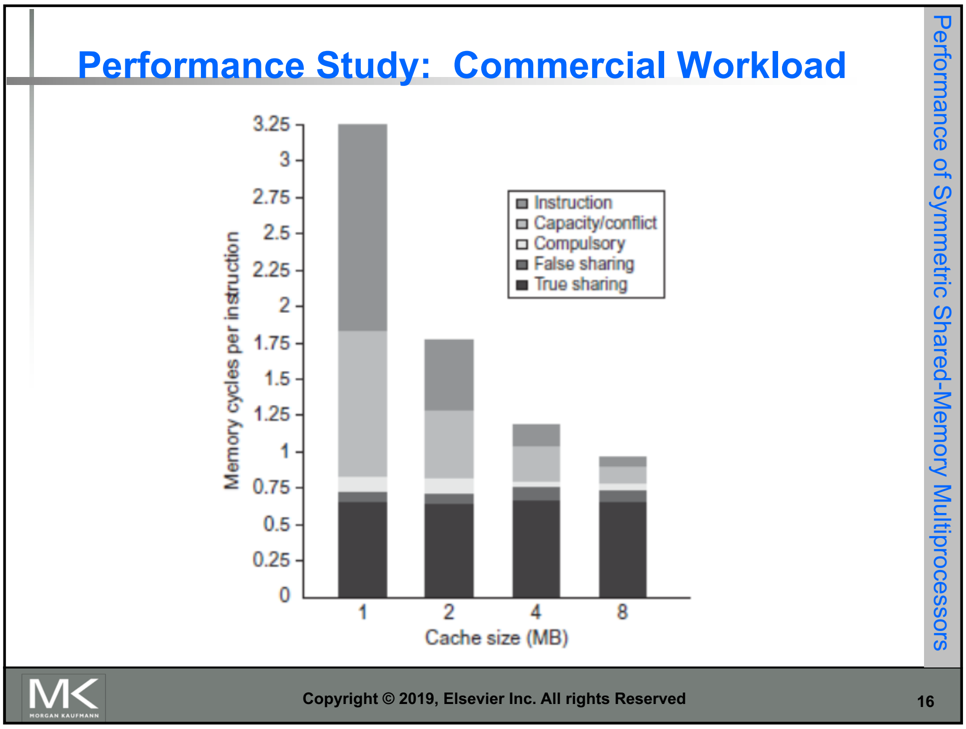
Performance

- Coherence influences cache miss rate
  - Coherence misses
    - True sharing misses
      - Write to shared block (transmission of invalidation)
      - Read an invalidated block
    - False sharing misses
      - Read an unmodified word in an invalidated block

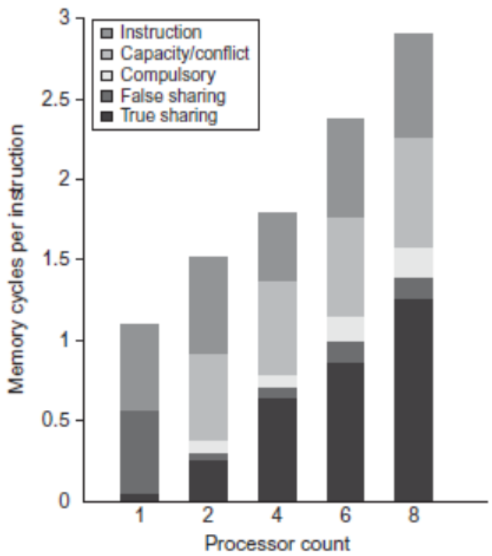
Performance of Symmetric Shared-Memory Multiprocessors

 14


Performance Study: Commercial Workload
Performance of Symmetric Shared-Memory Multiprocessors
Normalized execution time
100
90
80
70
60
50
40
30
20
10
0
1
2
4
8
L3 cache size (MB)
Idle
PAL code
Memory access
L2/L3 cache access
Instruction execution
Copyright © 2019, Elsevier Inc. All rights Reserved
15


Performance Study: Commercial Workload
Performance of Symmetric Shared-Memory Multiprocessors
Memory cycles per instruction
3.25
3
2.75
2.5
2.25
2
1.75
1.5
1.25
1
0.75
0.5
0.25
0
1
2
4
8
Cache size (MB)
Instruction
Capacity/conflict
Compulsory
False sharing
True sharing
Copyright © 2019, Elsevier Inc. All rights Reserved
16

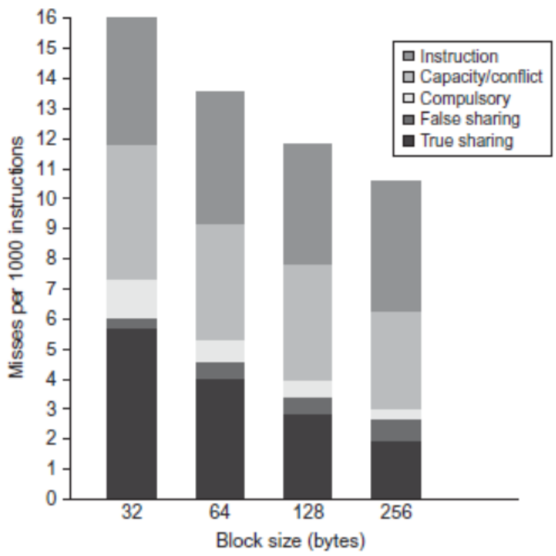
Performance Study: Commercial Workload



Performance of Symmetric Shared-Memory Multiprocessors

 17

Performance Study: Commercial Workload



Performance of Symmetric Shared-Memory Multiprocessors

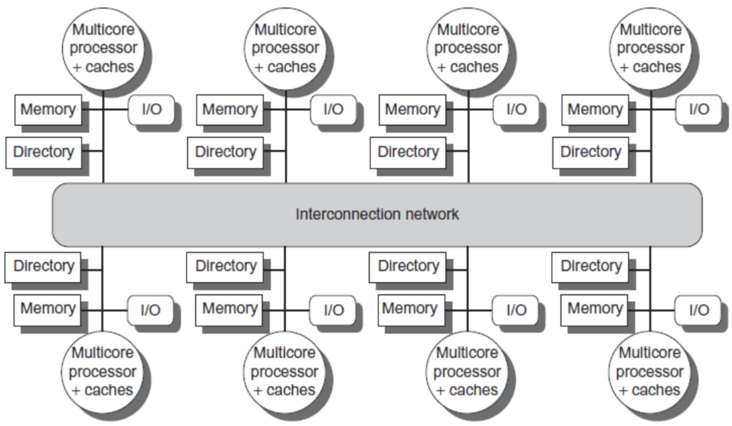
 18

Directory Protocols

- Snooping schemes require communication among all caches on every cache miss
  - Limits scalability
  - Another approach: Use centralized directory to keep track of every block
    - Which caches have each block
    - Dirty status of each block
- Implement in shared L3 cache
  - Keep bit vector of size = # cores for each block in L3
  - Not scalable beyond shared L3

Directory Protocols

- Alternative approach:
  - Distribute memory



Directory Protocols

Distributed Shared Memory and Directory-Based Coherence

- For each block, maintain state:
  - Shared
    - One or more nodes have the block cached, value in memory is up-to-date
    - Set of node IDs
  - Uncached
  - 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

 21

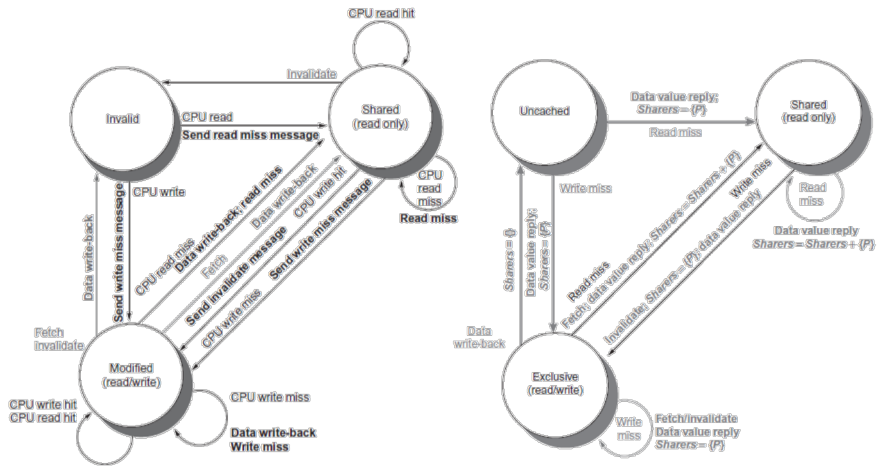
Messages

Distributed Shared Memory and Directory-Based Coherence

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 address A; request data and make P a read sharer.
Write miss	Local cache	Home directory	P, A	Node P has a write miss at address A; request data and make P the exclusive owner.
Invalidate	Local cache	Home directory	A	Request to send invalidates to all remote caches that are caching the block at address A.
Invalidate	Home directory	Remote cache	A	Invalidate a shared copy of data at address A.
Fetch	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared.
Fetch/ invalidate	Home directory	Remote cache	A	Fetch the block at address 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.

 22

Directory Protocols



Distributed Shared Memory and Directory-Based Coherence

 23

Directory Protocols

- 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

Distributed Shared Memory and Directory-Based Coherence

 24

Directory Protocols

- 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

Distributed Shared Memory and Directory-Based Coherence

Synchronization

- Basic building blocks:
  - Atomic exchange
    - Swaps register with memory location
  - Test-and-set
    - Sets under condition
  - Fetch-and-increment
    - Reads original value from memory and increments it in memory
  - Requires memory read and write in uninterruptable instruction

  - RISC-V: load reserved/store conditional
    - If the contents of the memory location specified by the load linked are changed before the store conditional to the same address, the store conditional fails

Synchronization

Implementing Locks

Synchronization

- Atomic exchange (EXCH):

```
try:    mov x3,x4        ;mov exchange value
        lr x2,x1         ;load reserved from
        sc x3,0(x1)      ;store conditional
        bnez x3,try      ;branch store fails
        mov x4,x2        ;put load value in x4?
```

- Atomic increment:

```
try:    lr x2,x1         ;load reserved 0(x1)
        addi x3,x2,1     ;increment
        sc x3,0(x1)      ;store conditional
        bnez x3,try      ;branch store fails
```

MK MORGAN KAUFMANN

 27

Implementing Locks

Synchronization

- Lock (no cache coherence)

```
        addi x2,R0,#1
lockit: EXCH x2,0(x1)        ;atomic exchange
        bnez x2,locket       ;already locked?
```

- Lock (cache coherence):

```
lockit: ld x2,0(x1)          ;load of lock
        bnez x2,locket       ;not available-spin
        addi x2,R0,#1        ;load locked value
        EXCH x2,0(x1)        ;swap
        bnez x2,locket       ;branch if lock wasn’t 0
```

MK MORGAN KAUFMANN

 28

Implementing Locks

- Advantage of this scheme: reduces memory traffic

Step	P0	P1	P2	Coherence state of lock at end of step	Bus/directory activity
1	Has lock	Begins spin, testing if lock = 0	Begins spin, testing if lock = 0	Shared	Cache misses for P1 and P2 satisfied in either order. Lock state becomes shared.
2	Set lock to 0	(Invalidate received)	(Invalidate received)	Exclusive (P0)	Write invalidate of lock variable from P0.
3		Cache miss	Cache miss	Shared	Bus/directory services P2 cache miss; write-back from P0; state shared.
4		(Waits while bus/ directory busy)	Lock = 0 test succeeds	Shared	Cache miss for P2 satisfied.
5		Lock = 0	Executes swap, gets cache miss	Shared	Cache miss for P1 satisfied.
6		Executes swap, gets cache miss	Completes swap: returns 0 and sets lock = 1	Exclusive (P2)	Bus/directory services P2 cache miss; generates invalidate; lock is exclusive.
7		Swap completes and returns 1, and sets lock = 1	Enter critical section	Exclusive (P1)	Bus/directory services P1 cache miss; sends invalidate and generates write-back from P2.
8		Spins, testing if lock = 0			None

Synchronization

 29

Models of Memory Consistency

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

- Should be impossible for both if-statements to be evaluated as true
  - Delayed write invalidate?
- Sequential consistency:
  - Result of execution should be the same as long as:
    - Accesses on each processor were kept in order
    - Accesses on different processors were arbitrarily interleaved

Models of Memory Consistency: An Introduction

 30

Implementing Locks

- To implement, delay completion of all memory accesses until all invalidations caused by the access are completed
  - Reduces performance!
- Alternatives:
  - Program-enforced synchronization to force write on processor to occur before read on the other processor
    - Requires synchronization object for A and another for B
      - "Unlock" after write
      - "Lock" after read

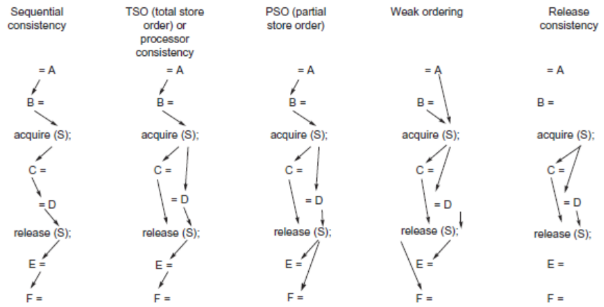
Relaxed Consistency Models

Models of Memory Consistency: An Introduction

- Rules:
  - X → Y
    - Operation X must complete before operation Y is done
    - Sequential consistency requires:
      - R → W, R → R, W → R, W → W
  - Relax W → R
    - "Total store ordering"
  - Relax W → W
    - "Partial store order"
  - Relax R → W and R → R
    - "Weak ordering" and "release consistency"

Relaxed Consistency Models

Model	Used in	Ordinary orderings	Synchronization orderings
Sequential consistency	Most machines as an optional mode	R→R, R→W, W→R, W→W	S→W, S→R, R→S, W→S, S→S
Total store order or processor consistency	IBMS/370, DEC VAX, SPARC	R→R, R→W, W→W	S→W, S→R, R→S, W→S, S→S
Partial store order	SPARC	R→R, R→W	S→W, S→R, R→S, W→S, S→S
Weak ordering	PowerPC		S→W, S→R, R→S, W→S, S→S
Release consistency	MIPS, RISC V, Armv8, C, and C++ specifications		$S_A \rightarrow W$, $S_A \rightarrow R$, $R \rightarrow S_R$, $W \rightarrow S_R$, $S_A \rightarrow S_A$, $S_A \rightarrow S_R$, $S_R \rightarrow S_A$, $S_R \rightarrow S_R$



Relaxed Consistency Models

- Consistency model is multiprocessor specific
- Programmers will often implement explicit synchronization
- Speculation gives much of the performance advantage of relaxed models with sequential consistency
  - Basic idea: if an invalidation arrives for a result that has not been committed, use speculation recovery

Fallacies and Pitfalls

Fallacies and Pitfalls

- Measuring performance of multiprocessors by linear speedup versus execution time
- Amdahl's Law doesn't apply to parallel computers
- Linear speedups are needed to make multiprocessors cost-effective
  - Doesn't consider cost of other system components
- Not developing the software to take advantage of, or optimize for, a multiprocessor architecture