Parallel Processing

- Uniprocessors (single core) come to an end
  - Slowing ability to extract ILP, increasing cost for ILP
  - Power consumption limits

1. Do many tasks at once: design for task parallelism
2. Shift to cloud, data intensive which are highly parallel
3. Improvement in parallel processing architecture
4. Benefits from easier replication (e.g., verification)

Task level parallelism with
Multiple Instruction, Multiple Data (MIMD)

Parallel Processing

- Multithreaded programs
  - Thread is unit of parallelism – it's a body of code
  - Multiple threads work together to do work
  - Threads share same address space
  - Lightweight communication, synchronization
- Multiprogrammed or request parallelism
  - Independent programs or requests
  - Do not communicate or synchronize
  - Less emphasis on comm/synch
  - More emphasis on contention among multiple programs
- ***Shared address vs. separate address spaces***

Parallel Processing

Multicore: Multiple CPUs on same chip die

Multiprocessor: Multiple processors in same box

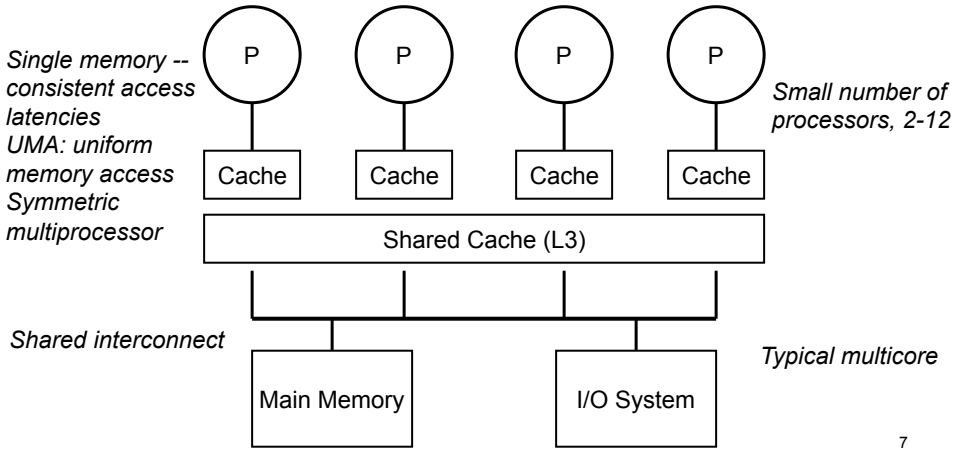
- Multiprocessor uses Multicore processors

- Symmetric (shared-memory) multiprocessors
- Distributed shared memory multiprocessor

6

Centralized Shared-Memory Architectures

Multiple processors sharing a single memory



Single memory -- consistent access latencies
UMA: uniform memory access
Symmetric multiprocessor

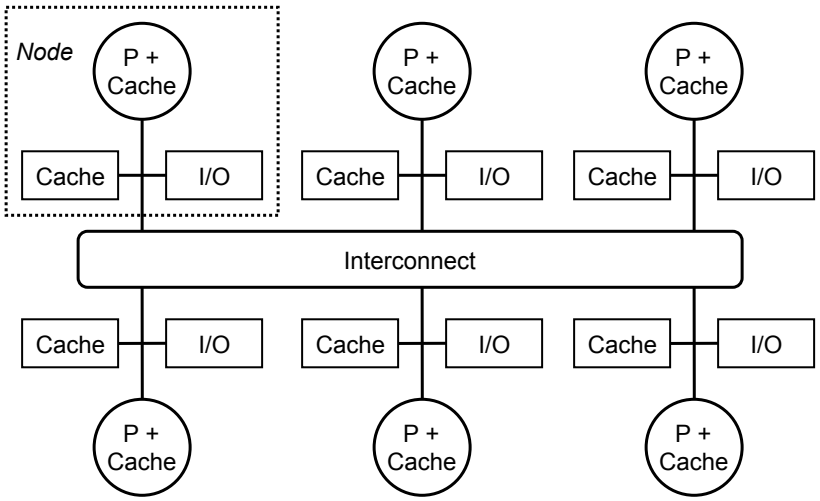
Small number of processors, 2-12

Shared interconnect

Typical multicore

7

Distributed Memory Architectures



8

Individual interconnected PEs with memory at each node - network connected MPs

Distributed Memory Architectures

- Shared memory systems don't scale well (why?)
- More processors, more bandwidth demands
- Distributed memory system
  - Typically high bandwidth interconnect
  - Cost-effective scaling of memory bandwidth
    - Assuming most accesses to local memory
    - Limited node-to-node communication & synchronization
  - Lower latency to local memory
    - Don't have to go "across bus" to shared memory
  - Communication among nodes is more complex
  - Communication has higher latency

9

Communication with Distributed Memory Architectures

- Distributed shared-memory
  - One logical memory distributed among physical memories
  - I.e., address space is shared (same *shared address* on two processors refers to the *same* location)
  - Implicit shared communication (via shared address space)
  - NUMA: Non-uniform memory access (why?)
- Multicomputers
  - Separate private address spaces for each PE
  - Same address on two processors: two different locations
  - Explicit communication (message passing)
  - Libraries for standard communication primitives (e.g., MPI)

10

Communication Performance

- Communication bandwidth (end-to-end)
  - Typically less than what the hardware can provide
  - Occupancy: resources are occupied during communication, preventing send/receive of other messages
- Communication latency
  - Overhead + time of flight + transport latency
  - Hiding latency is good!
    - » Ties up resources or the processor has to wait
  - Overhead can include occupancy
    - » May also include other items: protection provided by OS

11

Communication Performance

- Latency Hiding
  - Overlap communication with other communications or computations
  - Can be difficult to exploit and application dependent
- Flexible communication mechanisms
  - Perform well with
    - » Smaller and larger transmissions
    - » Irregular and regular communication patterns
  - I.e., not overly optimized
  - But…. May be able to improve communication performance if optimized for specific patterns (e.g., interconnection topology)

12

Communication Comparison

Shared Memory

- Compatibility, well understood
- Ease of programming for complex communication (just do it!)
- Better for smaller communications
  Protection implemented in the HW rather than in the OS
- Hardware-controlled caching
  Automatic caching of shared and private data
- Easy to implement message passing on top of shared memory since it's just a memory copy

Message Passing

- Simpler hardware (coherence)
- Explicit communication
  Have to pay attention! And get it right (often not easy, though…)
- Shared memory can be built on top of message passing but the cost is very high (every access becomes a message!)

13

Cache Coherence

Multilevel caches included with each processor
Private and shared data
Cache Coherence problem

Event	P1's Cache	P2's Cache	Memory
			1
P1: LD r1,[A]	1		1
P2: LD r1,[A]	1	1	1
P1: ADD r1,1,r1	1	1	1
P1: ST r1,[A]	2	1	2

22

Cache Coherence

Coherent if:

1: Write by processor P to X
Read by processor P of X
No intervening write
Returns most recent value

Preserves program order, true even of uniprocessors

2: Write by processor P1
Read by processor P2
Returns most recent value
if operations separated by enough time

Notion of coherency- get the most recent value

3: Writes to same location are serialized
I.e., writes seen by all processors in the same order

Ensures a value is not held indefinitely (if seen in different order)

23

Coherence Mechanisms

- Migration
  - Data moved to a local cache where it can be accessed locally
  - Reduces latency to shared data that is allocated remotely
- Replication
  - Copies of shared data that can be read by multiple processors
  - Reduces latency and contention for shared item
- Directory-based - Centralized directory tracks current location of data
- Snooping - State of blocks kept at local caches by watching interconnect (bus) transactions

24

Coherence Protocols

- Write invalidate
  - Only one processor has exclusive write access
  - No other readable/writable copies of to-be-written data exist
  - On write, invalidate all copies
  - Modify data, when other processors use it, they miss and get the new data
- Write broadcast
  - On a write, broadcast the updated value to all caches holding a copy of the data
  - Bandwidth requirements - keep track of whether a word is shared or not so unnecessary broadcasts are avoided

25

Example: Invalidate

Processor Activity	Bus Activity	Cache Contents for CPU A	Cache Contents for CPU B	Memory Contents for location X
				0

26

Example: Invalidate

Processor Activity	Bus Activity	Cache Contents for CPU A	Cache Contents for CPU B	Memory Contents for location X
				0
CPU A Reads X	Cache Miss for X	0		0

27

Example: Invalidate

Processor Activity	Bus Activity	Cache Contents for CPU A	Cache Contents for CPU B	Memory Contents for location X
				0
CPU A Reads X	Cache Miss for X	0		0
CPU B Reads X	Cache Miss for X	0	0	0

28

Example: Invalidate

Processor Activity	Bus Activity	Cache Contents for CPU A	Cache Contents for CPU B	Memory Contents for location X
				0
CPU A Reads X	Cache Miss for X	0		0
CPU B Reads X	Cache Miss for X	0	0	0
CPU A writes 1 to X	Invalidation for X	1		0

29

Example: Invalidate

Processor Activity	Bus Activity	Cache Contents for CPU A	Cache Contents for CPU B	Memory Contents for location X
				0
CPU A Reads X	Cache Miss for X	0		0
CPU B Reads X	Cache Miss for X	0	0	0
CPU A writes 1 to X	Invalidation for X	1		0
CPU B Reads X	Cache Miss for X	1	1	1

30

Example: Write Update

Processor Activity	Bus Activity	Cache Contents for CPU A	Cache Contents for CPU B	Memory Contents for location X
				0
CPU A Reads X	Cache Miss for X	0		0
CPU B Reads X	Cache Miss for X	0	0	0

31

Example: Write Update

Processor Activity	Bus Activity	Cache Contents for CPU A	Cache Contents for CPU B	Memory Contents for location X
				0
CPU A Reads X	Cache Miss for X	0		0
CPU B Reads X	Cache Miss for X	0	0	0
CPU A writes 1 to X	Write update for X	1	1	1

32

Example: Write Update

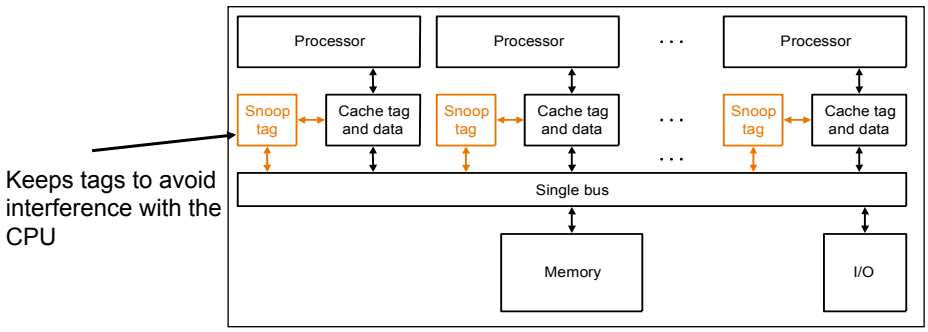
Processor Activity	Bus Activity	Cache Contents for CPU A	Cache Contents for CPU B	Memory Contents for location X
				0
CPU A Reads X	Cache Miss for X	0		0
CPU B Reads X	Cache Miss for X	0	0	0
CPU A writes 1 to X	Write update for X	1	1	1
CPU B Reads X		1	1	1

33

Invalidate vs. Broadcast

- Multiple writes to same word (with no intervening write): multiple write broadcasts but a single write invalidate
- Cache line blocks: multiple writes to block require multiple broadcasts but only one invalidate when block first written. Broadcast works only on individual words as opposed to blocks.
- Delay between seeing a write: usually less in write broadcast since data is immediately updated in a reader s cache

Snooping Protocols



Keeps tags to avoid interference with the CPU

- Each processor monitors the activity on the bus
- Dealing with *write through* is simpler than dealing with *write back*
- In WB, on a read miss, all caches check to see if they have a copy of the requested block. If yes, they supply the data (will see how).
- In WB, on a write miss, all caches check to see if they have a copy of the requested data. Yes: invalidate the local copy or update it with the new value.

Snoopy MSI Protocol

- Invalidation protocol
- Each block of memory is in one state:
  - Clean in all caches and up-to-date in memory (Read-Only),
  - Dirty in exactly one cache (Read/Write), OR
  - Not in any caches
- Each cache block is in one state:
  - Shared : block can be read (clean, read-only)
  - Modified: cache has only copy, its writeable, and dirty
  - Invalid : block contains no data
- Read misses: cause all caches to snoop bus
- Writes to clean blocks are treated as misses -- invalidates all other caches

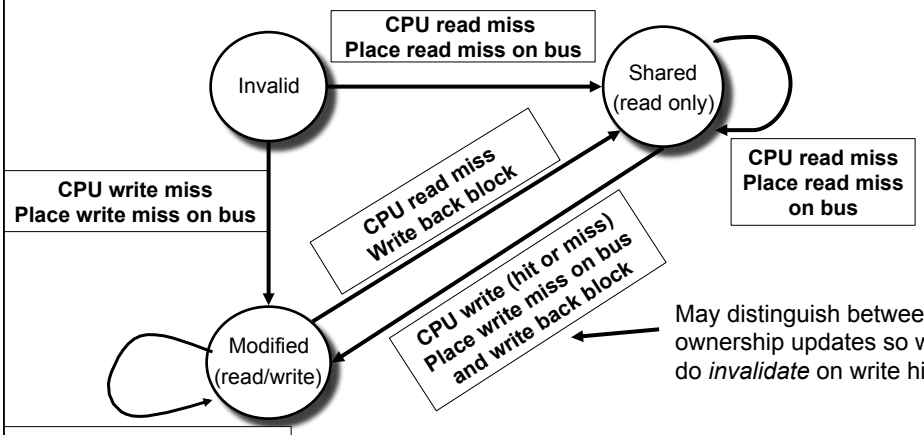
36

Snoopy MSI Protocol

- Interconnection (bus) based systems
  - Acquire bus to do invalidation - write not complete until invalidated (and have the bus)
  - Causes serialization of writes
- Finite state machine to maintain coherence
  - Status bits for each cache line (protocol state)
  - 2 parts to FSM
    - » CPU activity on one processor (CPU events)
    - » other processors see activity (bus events)
- Only one processor has write access at a time
- All processors can have read access together

37

Snoopy State Machine (CPU Events)

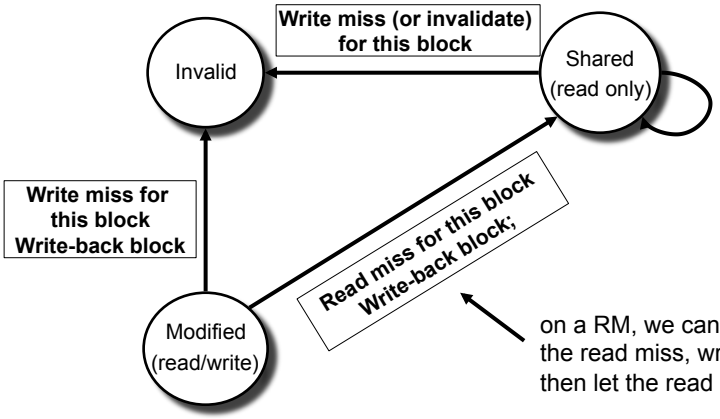


May distinguish between ownership updates so we do *invalidate* on write hit

Note: A read hit does not change the state.

38

Snoopy State Machine (Bus Events)



on a RM, we can intercept the read miss, write the block, then let the read proceed

39

What Happens When...

- Read miss - always go to SHARED
  - on a RM, other processors may be in INVALID, SHARED, or MODIFIED
    - » INVLAID: No action, stay in INVALID
    - » SHARED: No action, stay in SHARED
    - » MODIFIED: Exclusive processor does writeback and goes to SHARED
  - this processor goes to shared

40

What Happens When...

- Write miss - get exclusive access, invalidate other copies
- on a WM, we will always put address on the bus so other processors can go to INVALID with a possible write back
- if this processor:
  - INVALID -> MODIFIED, put address on bus
  - MODIFIED-> MODIFIED, writeback replaced line
  - SHARED -> MODIFIED, put address on bus

41

What Happens When...

- Write hit - get exclusive access
  - if processor:
    - » in SHARED -> MODIFIED, invalidate
    - » in EXCLUSIVE -> MODIFIED, no action
  - this processor has exclusive access, so no other processor will do anything
- Read hit - stay in state; no action required (common case)

42

Example

Assume: A1 and A2 map to same cache block B, initial cache state is invalid

Event	In P1's cache	In P2's cache
	B = invalid	B = invalid

43

Example

Assume: A1 and A2 map to same cache block B, initial cache state is invalid

Event	In P1's cache	In P2's cache
	B = invalid	B = invalid
P1 writes 10 to A1		

44

Example

Assume: A1 and A2 map to same cache block B, initial cache state is invalid

Event	In P1's cache	In P2's cache
	B = invalid	B = invalid
P1 writes 10 to A1	A1 = 10 (modified)	B = invalid

45

Example

Assume: A1 and A2 map to same cache block B, initial cache state is invalid

Event	In P1's cache	In P2's cache
	B = invalid	B = invalid
P1 writes 10 to A1	A1 = 10 (modified)	B = invalid
P1 reads A1		

46

Example

Assume: A1 and A2 map to same cache block B, initial cache state is invalid

Event	In P1's cache	In P2's cache
	B = invalid	B = invalid
P1 writes 10 to A1	A1 = 10 (modified)	B = invalid
P1 reads A1 (RH)	A1 = 10 (modified)	B = invalid

47

Example

Assume: A1 and A2 map to same cache block B, initial cache state is invalid

Event	In P1's cache	In P2's cache
	B = invalid	B = invalid
P1 writes 10 to A1	A1 = 10 (modified)	B = invalid
P1 reads A1 (RH)	A1 = 10 (modified)	B = invalid
P2 reads A1		

48

Example

Assume: A1 and A2 map to same cache block B, initial cache state is invalid

Event	In P1's cache	In P2's cache
	B = invalid	B = invalid
P1 writes 10 to A1	A1 = 10 (modified)	B = invalid
P1 reads A1 (RH)	A1 = 10 (modified)	B = invalid
P2 reads A1 (RM)	A1 = 10 (shared)	A1 = 10 (shared)

49

Example

Assume: A1 and A2 map to same cache block B, initial cache state is invalid

Event	In P1's cache	In P2's cache
	B = invalid	B = invalid
P1 writes 10 to A1	A1 = 10 (modified)	B = invalid
P1 reads A1 (RH)	A1 = 10 (modified)	B = invalid
P2 reads A1 (RM)	A1 = 10 (shared)	A1 = 10 (shared)
P2 write 20 to A1		

50

Example

Assume: A1 and A2 map to same cache block B, initial cache state is invalid

Event	In P1's cache	In P2's cache
	B = invalid	B = invalid
P1 writes 10 to A1	A1 = 10 (modified)	B = invalid
P1 reads A1 (RH)	A1 = 10 (modified)	B = invalid
P2 reads A1 (RM)	A1 = 10 (shared)	A1 = 10 (shared)
P2 write 20 to A1 (WH)	B = invalid	A1 = 20 (modified)

51

Example

Assume: A1 and A2 map to same cache block B, initial cache state is invalid

Event	In P1's cache	In P2's cache
	B = invalid	B = invalid
P1 writes 10 to A1	A1 = 10 (modified)	B = invalid
P1 reads A1 (RH)	A1 = 10 (modified)	B = invalid
P2 reads A1 (RM)	A1 = 10 (shared)	A1 = 10 (shared)
P2 write 20 to A1 (WH)	B = invalid	A1 = 20 (modified)
P2 writes 40 to A2		

52

Example

Assume: A1 and A2 map to same cache block B, initial cache state is invalid

Event	In P1's cache	In P2's cache
	B = invalid	B = invalid
P1 writes 10 to A1	A1 = 10 (modified)	B = invalid
P1 reads A1 (RH)	A1 = 10 (modified)	B = invalid
P2 reads A1 (RM)	A1 = 10 (shared)	A1 = 10 (shared)
P2 write 20 to A1 (WH)	B = invalid	A1 = 20 (modified)
P2 writes 40 to A2 (WM)	B = invalid	

53

Example

Assume: A1 and A2 map to same cache block B, initial cache state is invalid

Event	In P1's cache	In P2's cache
	B = invalid	B = invalid
P1 writes 10 to A1	A1 = 10 (modified)	B = invalid
P1 reads A1 (RH)	A1 = 10 (modified)	B = invalid
P2 reads A1 (RM)	A1 = 10 (shared)	A1 = 10 (shared)
P2 write 20 to A1 (WH)	B = invalid	A1 = 20 (modified)
P2 writes 40 to A2 (WM)	B = invalid	A2 = 40 (modified)

54

MESI Protocol

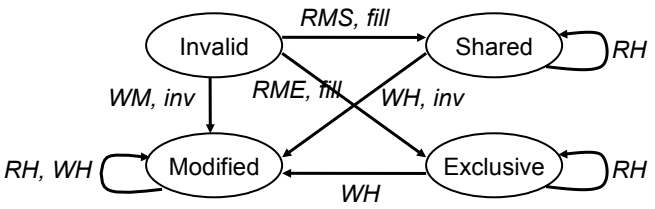
- Extend 3 state protocol to have a "modified" state
  - **M**odified: Line has been modified (different from main memory; no other copy)
  - **E**xclusive: line is same as in main memory but we have exclusive access (not shared)
  - **S**hared: same as in memory and present in other caches
  - **I**nvalid: line is not valid
- Shared in 3 state protocol is *shared* & *exclusive*
  - Avoids invalidate when no other processor has a copy of data and we want to do a write of it
- Extra signaling to indicate whether other caches have a copy or not (I.e., shared or exclusive?)

55

MESI State Diagram

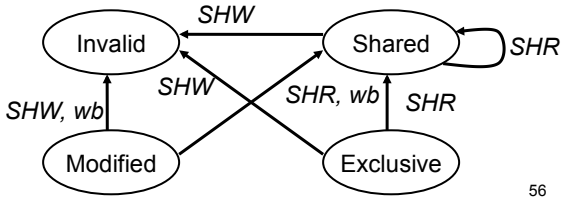
Processor-View

RMS - RM shared
RME - RM exclusive
inv - invalidate
fill - cache line fill
NOTE: not all events are shown (e.g., RM or WM in exclusive)



Bus-View

SHR - shared read
SHW - shared write
wb - write back



56