Parallel Architectures

Chapter 4:
Caches and cache coherency protocols

Lecturer: *Paul H J Kelly*

Autumn 1997

Department of Computing
Imperial College

Room 423
email: phjk@doc.ic.ac.uk

Course web site:
http://www-ala.doc.ic.ac.uk/~phjk/phjk-ParallelArchitectures.html

CHAPTER 4

4.1 Memory price/performance

- Dynamic RAM (DRAM):
  - Square array, one transistor per bit, data stored as charge on capacitor
  - Very low power
  - Discharge to read, so must write back — so cycle time > access time
  - Charge slowly decays so needs periodic refresh
  - Address passed in two halves:
    - Row address (RAS) transfers data from a whole row into an on-chip buffer
    - Column address (CAS) selects required bit from buffer
    - Can issue one RAS then use many different CAS's to access the row buffer
  - Access time: ca.50ns (10-50 cycles)
  - Cycle time: ca.90ns
  - Commonly 64Mbit/chip or more
  - Various higher-performance interfaces available (SDRAM, RamBus, VDRAM)

Memory price/perf - cont'd

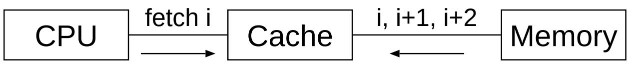
- Static RAM (SRAM)
  - 4–6 transistors/bit, VLSI die area 4–6× larger
  - Uses power just to hold data
  - 4–8× (or more) faster
  - 4–8× (or more) more expensive

(H&P pp.9,41,429,437)

4.2 Memory Hierarchies

- Use a small amount of fast, expensive memory, backed up by a large amount of slower, cheaper memory which is large enough to store all data
- If an instruction refers to a datum which is not in the cache, it is automatically copied from the slow memory into the cache
- If most accesses can be satisfied by the cache, the average latency of memory accesses will be close to the cycle time of the cache
- We aim to achieve the performance of SRAM (expensive, fast) by using DRAM (cheaper, slower)

Why Might it Work?



- Loops: a small set of instructions is repeatedly executed. Instructions that were accessed in the recent past will hopefully still be in the cache when they are next required
- Procedure call stacks, arrays...

Temporal locality: recently referenced items are likely to be referenced again soon, e.g. loops, stacks, temporary variables

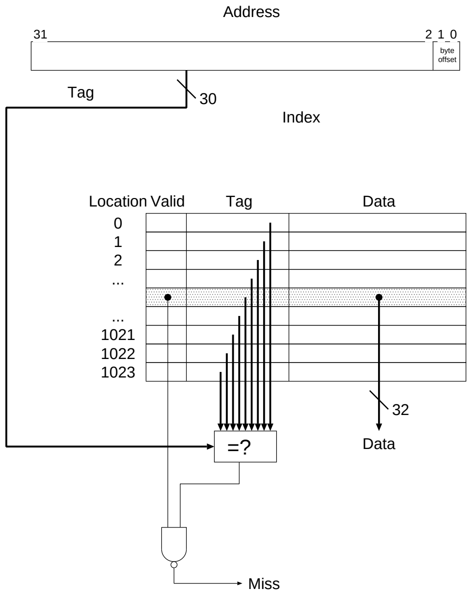
Spatial locality: programs tend to access items whose addresses are near one another, e.g. arrays, code segments, stacks

Hit ratio: the ratio of the number of cache hits to the number of cache accesses

Fundamental definitions

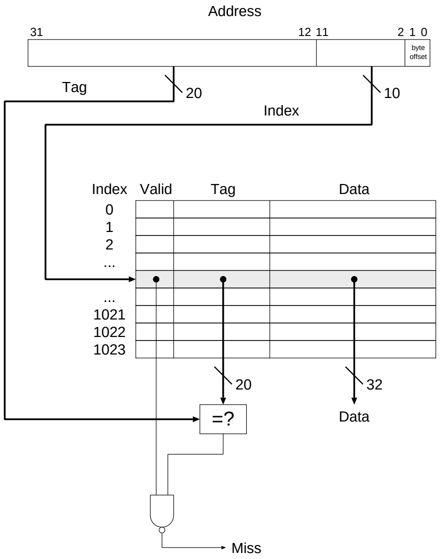
- **Block:** a $B \times N$-word cache provides fast storage for up to B blocks, each consisting of N words
- **Tag:** Each block held in the cache must be tagged with its main memory address
- **Read Hit:** If tag matches address, the data is supplied to the processor
- **Read Miss:** no match: a block must be fetched from memory
- **Allocate:** when a new block is fetched, it is stored in the cache
- **Replacement:** to make space for a new block to be allocated in the cache, it may be necessary to displace an existing block.

4.3 Fully-associative caches



- In a fully associative cache, each main memory block can be allocated to any location in the cache
- To check whether a given block is in the cache, must compare address with every tag

4.4 Direct-Mapped Caches



- In a direct-mapped cache, each main memory block can be allocated to just one location in the cache
- To check whether a given block is in the cache, compare address with just the tag for its location in the cache

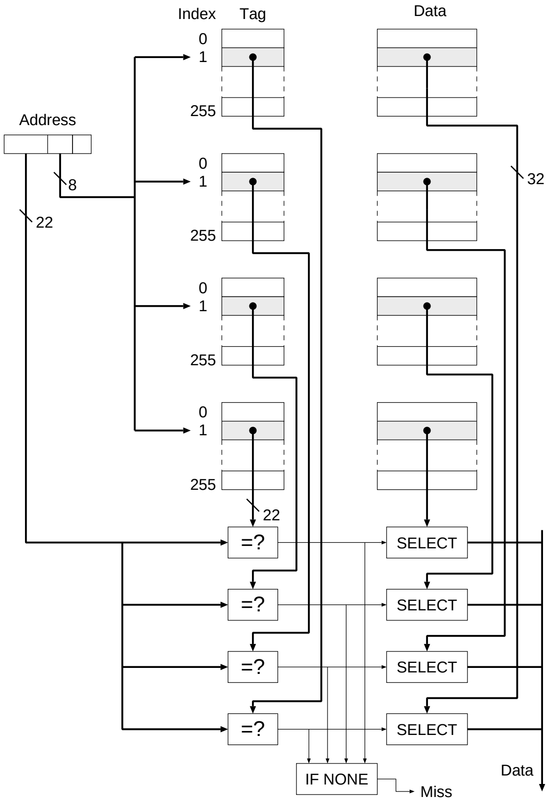
Cache Design Issues

- *Block placement:* where can a block be placed in the cache?
- *Block replacement:* which block should be replaced on a miss?
- *Write strategy:* what happens on a write?
- *Cache parameters:* e.g. block size, cache size, associativity

4.5 Block Placement

- *Direct-mapped:* block is limited to one location: only one comparator per cache
- *Set-associative:* block can be stored in any of n locations: n comparators required
- *Fully-associative:* block can be stored in any location

A 4-Way Set Associative Cache



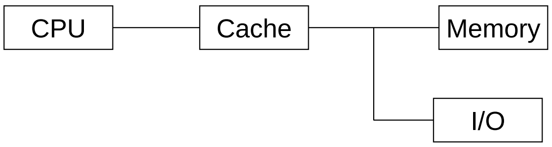
- *Fully-associative:* block can be stored in any location: each block needs a comparator
- *Direct-mapped:* block is limited to one location: only one comparator per cache
- Slower than direct-mapped (why?)
- Omitted valid bits and replacement stuff

4.6 Block Replacement

- On a cache miss, a block must be expelled before the new block is brought in from memory (unless the valid bit indicates that the cache block is empty)
- No choice for a direct-mapped cache
- Objective: to retain blocks which will be used again soon
  - *Optimal* (OPT): replace the block which will not be used for the longest time
  - *Random*
  - *Least-recently used* (LRU): replace the block which has been unused for the longest time
- LRU uses the past as an estimate of the future, i.e. the corollary of temporal locality: if recently used blocks are likely to be used again, the best candidate for replacement is the least recently used block

- **Q:** what is relationship between LRU and OPT?
- **Q:** how could LRU be implemented? Cost?
- Tradeoff between implementation cost and performance
- **Q:** how could LRU be approximated?
- See H&P pp.379 for comparison of LRU and random replacement
  - Random is nearly as good as LRU, especially for larger caches
  - Random is considerably easier to implement

4.7 Write Policies



Write-Through

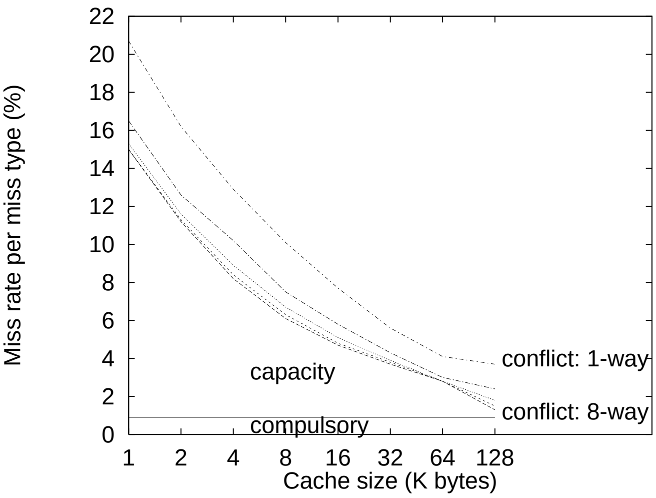
- Data is written to the block in the cache and the block in the lower-level memory
- Cycle time is that of memory
- Memory is always *coherent*

Write-Back

- Data is only written to the cache
- Modified blocks are written to memory only when replaced
- Cycle time of a hit is that of the cache
- Uses *dirty* bits to mark blocks which need to be written back when replaced

4.8 Trace-Based Cache Analysis

- Many design choices, e.g. degree of associativity, cache size, block size, replacement policy, write policy
- Most represent cost/performance tradeoffs. Cache simulators are used by designers to study these tradeoffs, and how cache parameters interact
- The effect of varying cache size and associativity is shown, based on a trace of a multiprogramming workload on a VAX with 32-byte blocks and LRU replacement (cf H&P pp 392)



Compulsory: the first access to a block requires it to be brought in from a higher level

Capacity: occurs when a new block is brought into a full cache

Conflict: occurs when more than one block maps to the same cache block: blocks must be discarded and later retrieved

(cf H&P pp.392)

Conclusions of Trace-Based Analysis

- Compulsory misses are constant
- 2:1 rule of thumb: miss rate of a direct-mapped cache of size $N \approx$ miss rate of a 2-way set-associative cache of size $N/2$
- Thrashing occurs when the cache is too small to hold the *working set* of the program: the memory system runs at the speed of memory

The DINERO cache simulator and some traces can be found on DoC Unix machines in `/homes/phjk/ParallelArchitectures94/HennessyAndPattersonSoftware/dinero`

Try: `zcat ../traces/benchmarks/tex.din.Z | bin/dinero -b32 -i8192 -d8192 -a1`

Notes...

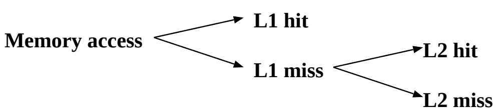
- **Q:** how would miss-rate vary with block size?
- **Q:** how might miss penalty for large blocks be reduced?
- **Q:** what happens with a large block size, when a write miss occurs?
- **Q:** Should the tag be the virtual address or the physical address?
- **Q:** Should the second-level cache always contain everything held in the first-level cache (the "multi-level inclusion" property)
- **Q:** How could one restructure a program to get better performance with a small cache?
- **Q:** What kind of program would have a very high rate of conflict misses with a two-way set-associative cache?

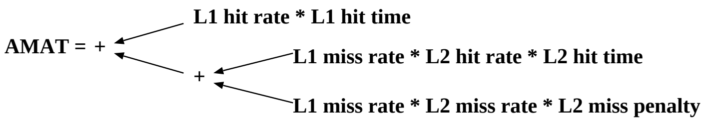
4.9 Average memory access time

The bottom line in memory system design is the average memory access time, AMAT. For a simple cache:

Average memory access time
  = Hit time + Miss rate × Miss penalty

For a machine with a secondary cache:





AMAT is, of course, only part of the complete performance equation:

CPU time
= (CPU execution clock cycles + Memory stall clock cycles)
× Clock cycle time

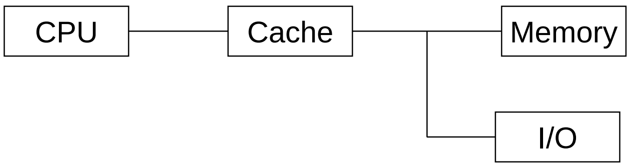
- In a highly-dynamic CPU, instruction scheduling may reduce stalls due to memory accesses
- Further stalls may occur due to structural hazards, eg simultaneous instruction and data accesses, or peripheral activities

Caches: Summary

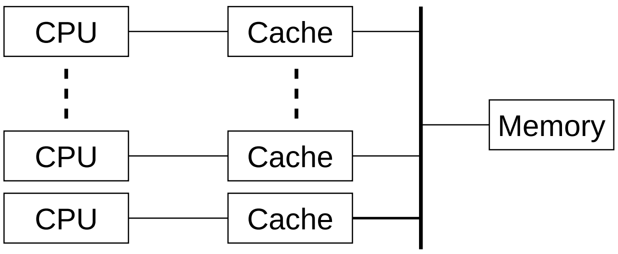
- SRAM is much faster than DRAM but too expensive and power-hungry for bulk storage
- Caches rely on spatial and temporal locality
- Design variables: capacity, associativity, block size, replacement policy and many more
- Write-through vs write-back
- Cache misses due to the three C's: Compulsory, Capacity, Conflict
- Average memory access time (AMAT) depends on applications's access pattern, hit time, miss rate, miss penalty

4.10 Shared-Memory Multiprocessors

Recall the write-back cache:



- Memory bus is not fully utilised
- Can add more processors and caches to form a parallel computer
- Each processors runs the same program, and communicates by reading and writing the shared-memory

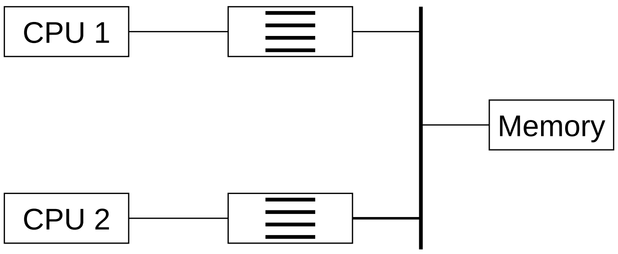


4.11 Cache Coherency

Multiple caches raise the *cache coherency* problem:

> the value returned by a read is the last value written to that location.

e.g. processors 1 and 2 both read a cache line. Processor 1 updates it. A read by processor 2 must return the new data



Solutions

- Writes cause invalidations or updates to be sent over the bus
- All caches "snoop" the bus for messages they need to act upon

Write Update

- Send a write message and the data over the bus
- Other caches update their copies
- Similar to *write-through*

Write Invalidate

- Send an invalidation message over the bus
- All other caches delete their copies
- Similar to *write-back*

Update *vs* invalidate

- Update:

  May reduce latency of subsequent remote reads (if any are made)

- Invalidate:

  May reduce network traffic (e.g. if same CPU writes to the line before remote nodes take copies of it)

4.12 The "Berkeley" Protocol

Cache line states:

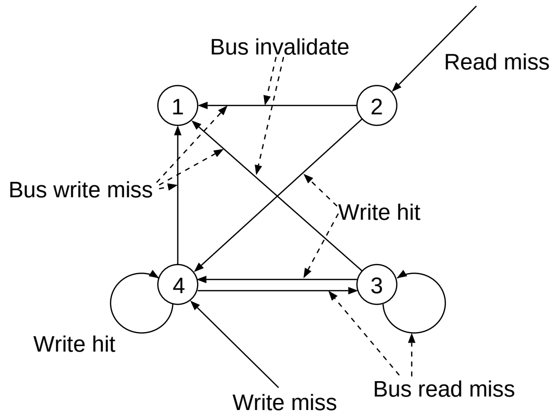
- INVALID
- VALID : clean, potentially shared, unowned
- SHARED-DIRTY : modified, possibly shared, owned
- DIRTY : modified, only copy, owned

Read miss: If another cache has the line in SHARED-DIRTY or DIRTY, it is supplied (changing state to SHARED-DIRTY). Otherwise the line comes from memory. The state of the line is set to VALID

Write hit: No action if line is DIRTY. If VALID or SHARED-DIRTY, an invalidation is sent, and the local state set to DIRTY

Write miss: Line comes from owner (as with read miss). All other copies set to INVALID, and line in requesting cache is set to DIRTY

State Transition Diagram



1. INVALID
2. VALID: clean, potentially shared, unowned
3. SHARED-DIRTY: modified, possibly shared, owned
4. DIRTY: modified, only copy, owned

4.13 Large-Scale Shared-Memory Multiprocessors

Bus inevitably becomes a bottleneck when many processors are used

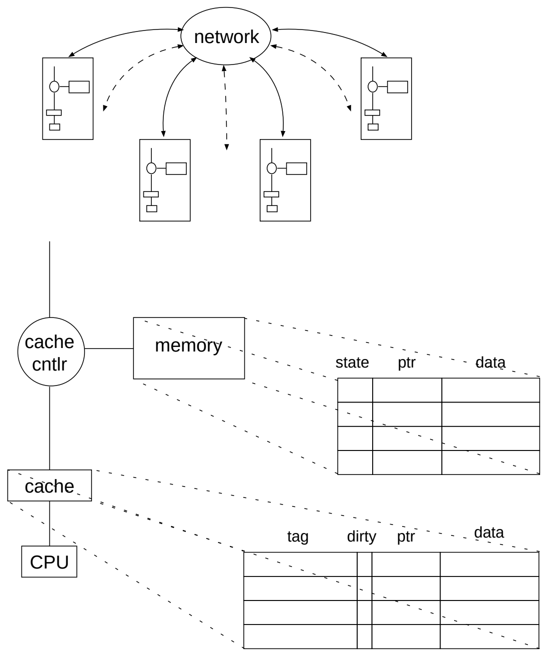
- Use a more general interconnection network, so snooping does not work
- DRAM memory is also distributed
- Each node allocates space from local DRAM
- Copies of remote data are made in cache

Major design issues:

- How to find and represent the "directory" of each line?
- How to find a copy of a line?

As a case study, we will look at S3.MP (Sun's Scalable Shared memory Multi-Processor, a CC-NUMA (cache-coherent non-uniform memory access) architecture

Case Study: S3.MP



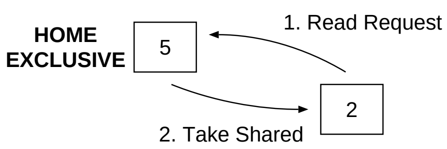
Protocol Basics

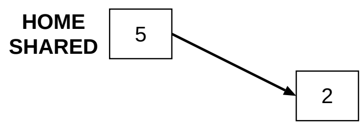
S3.MP uses distributed singly-linked sharing lists, with static homes

- Each line has a "home" node, which stores the root of the directory
- Requests are sent to the home node
- Home either has a copy of the line, or knows a node which does

Read Requests

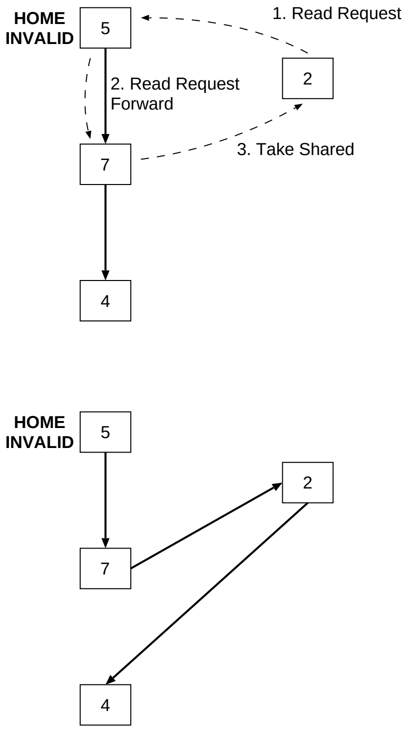
Simple case: initially only the home has the data:





Home replies with the data, creating a sharing chain containing just the reader

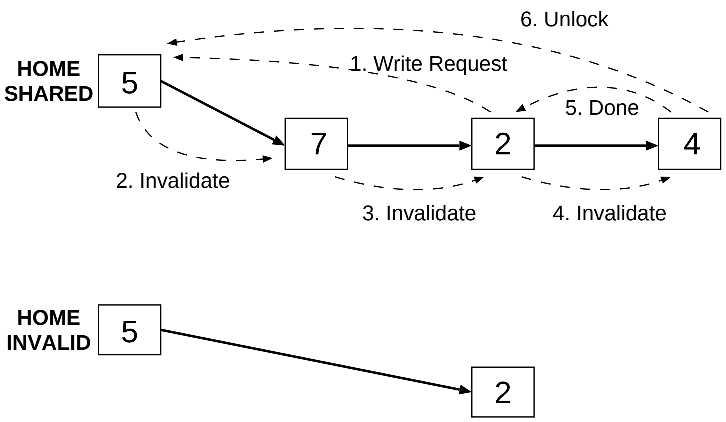
More interesting case: some other processor has the data:



Home passes request to first processor in chain, adding requester into the sharing list

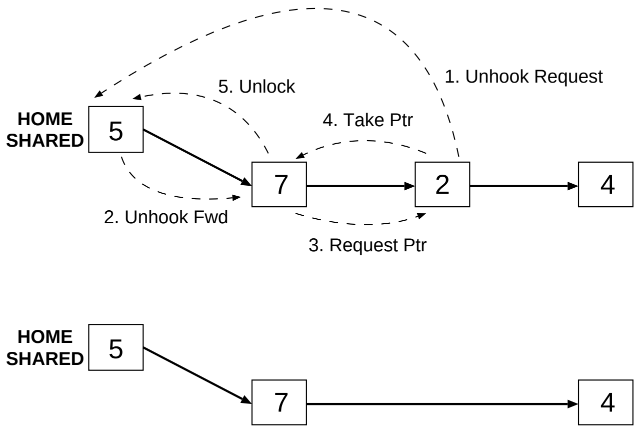
Writes

- If the line is exclusive (i.e. dirty bit is set) no message is required
- Else send a write-request to the home
- Home sends an invalidation message down the chain
- Each copy is invalidated (other than that of the requester)
- Final node in chain acknowledges the requester and the home
- Chain is locked for the duration of the invalidation



Replacements

- When a read or write requires a line to be copied into the cache from another node, an existing line may need to be *replaced*
- Must remove it from the sharing list
- Must not lose last copy of the line



Summary and Conclusions

- Caches are essential to gain the maximum performance from modern microprocessors
- The performance of a cache is close to that of SRAM but at the cost of DRAM
- Caches can be used to form the basis of a parallel computer
- Bus-based multiprocessors do not scale well: max$\approx$ 10 nodes
- Larger-scale shared-memory multiprocessors require more complicated networks and protocols

CC-NUMA is becoming popular since systems can be built from commodity components (chips, boards, OSs) and use existing software

e.g. HP/Convex, Sequent, Data General, CRAY, Sun, ICL

(Acknowledgements: Andrew Bennett wrote the first version of this chapter)