COMP 4611

Design and Analysis of Computer Architectures

Final Exam Review

Memory System

- Main memory generally uses Dynamic RAM (***DRAM***), which uses a single transistor to store a bit, but requires a periodic data refresh (~every 8 ms).
- Cache uses ***SRAM***: Static Random Access Memory
  - No refresh (6 transistors/bit vs. 1 transistor/bit for DRAM)
- *Size*: DRAM/SRAM *4-8*, *Cost & Performance*: SRAM/DRAM *8-16*
- Performance metrics
  - **Latency** is concern of cache
  - **Access time:** The time it takes between a memory access request and the time the requested information is available to cache/CPU.
  - **Cycle time:** The minimum time between unrelated requests to memory (greater than access time in DRAM to allow address lines to be stable)
  - **Memory bandwidth:** The maximum sustained data transfer rate between main memory and cache/CPU.

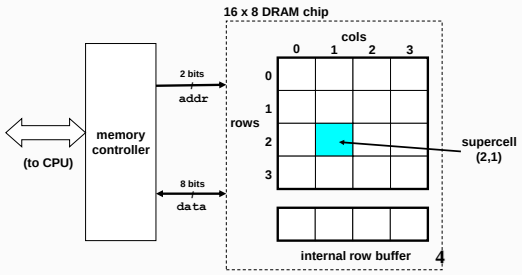
2

Memory Technology

- SRAM
  - Requires lower power to retain bit than DRAM
  - Requires 6 transistors/bit
- DRAM
  - Must be re-written after being read
  - Must also be periodically refeshed
    - Every ~ 8 ms
    - Each row can be refreshed simultaneously
  - One transistor/bit
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)

Conventional DRAM Organization

- d x w DRAM:
  - dw total bits organized as d supercells of size w bits

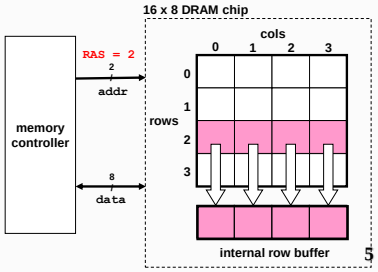


4

Reading DRAM Supercell (2,1)

Step 1(a): Row access strobe (RAS) selects row 2.

Step 1(b): Row 2 copied from DRAM array to row buffer.

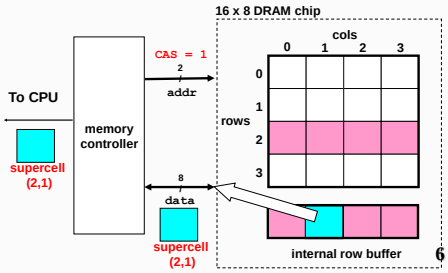


5

Reading DRAM Supercell (2,1)

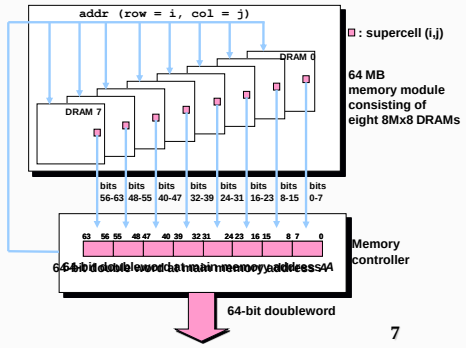
Step 2(a): Column access strobe (CAS) selects column 1.

Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.



6

Memory Modules



Memory Optimizations

- DDR (Double Data Rate):
  - DDR2
    - Lower power (2.5 V -> 1.8 V)
    - Higher clock rates (266 MHz, 333 MHz, 400 MHz)
  - DDR3
    - 1.5 V
    - 800 MHz
  - DDR4
    - 1-1.2 V
    - 1600 MHz
- GDDR5 is graphics memory based on DDR3

Memory Optimizations

- Graphics memory:
  - Achieve 2-5 X bandwidth per DRAM vs. DDR3
    - Wider interfaces (32 vs. 16 bit)
    - Higher clock rate
      - Possible because they are attached via soldering instead of socketted DIMM modules
- Reducing power in SDRAMs:
  - Lower voltage
  - Low power mode (ignores clock, continues to refresh)

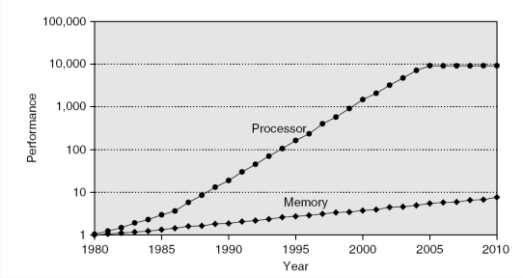
Flash Memory

- Type of EEPROM
- Must be erased (in blocks) before being overwritten
- Non-volatile
- Limited number of write cycles
- Cheaper than SDRAM, more expensive than disk
- Slower than SRAM, faster than disk

Memory Technology

- Amdahl:
  - Memory capacity should grow linearly with processor speed
  - Unfortunately, memory capacity and speed has not kept pace with processors
- Some optimizations:
  - Multiple accesses to same row
  - Synchronous DRAM
    - Added clock to DRAM interface
    - Burst mode with critical word first
  - Wider interfaces
  - Double data rate (DDR)
  - Multiple banks on each DRAM device

Memory Performance Gap



How to make memory system better?

- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory
- Solution: organize memory system into a hierarchy
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories, each containing a subset of the memory below it, proceed in steps up toward the processor
- Temporal and spatial locality insures that nearly all references can be found in smaller memories
  - Gives the illusion of a large, fast memory being presented to the processor

Memory Hierarchy Design

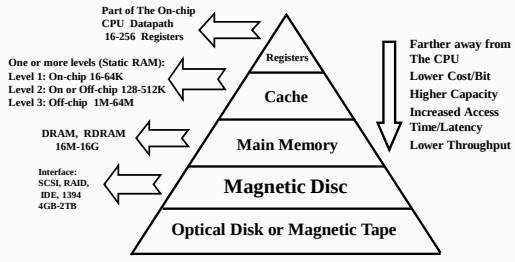
- Memory hierarchy design becomes more crucial with recent multi-core processors:
  - Aggregate peak bandwidth grows with # cores:
    - Intel Core i7 can generate two references per core per clock
    - Four cores and 3.2 GHz clock
      - 25.6 billion 64-bit data references/second +
      - 12.8 billion 128-bit instruction references
      - = 409.6 GB/s!
    - DRAM bandwidth is only 6% of this (25 GB/s)
    - Requires:
      - Multi-port, pipelined caches
      - Two levels of cache per core
      - Shared third-level cache on chip
- High-end microprocessors have >10 MB on-chip cache
  - Consumes large amount of area and power budget

Memory Hierarchy

- The idea is to build a memory subsystem that consists of:
  - Very small, very fast, very expensive memory "close" to the processor.
  - Larger, slower, but more affordable memory "further away" from the processor.
  - Hence, provide the appearance of virtually unlimited memory while minimizing delays to the processor.
- The memory hierarchy is organized into levels of memory with the smaller, more expensive, and faster memory levels closer to the CPU: registers, then primary Cache Level (L_1), then additional secondary cache levels (L_2, L_3…), then main memory, then mass storage (virtual memory).

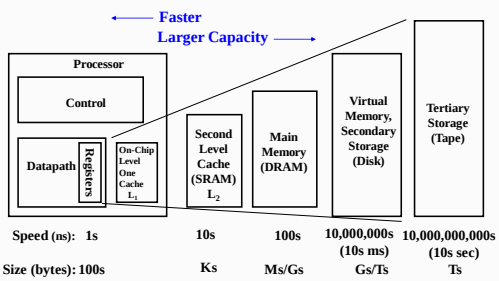
15

Levels of The Memory Hierarchy



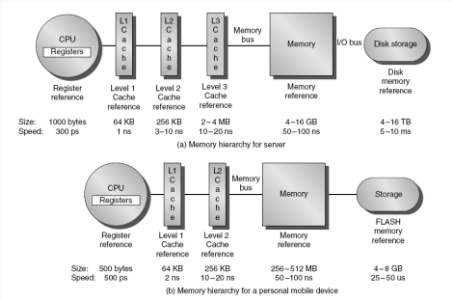
16

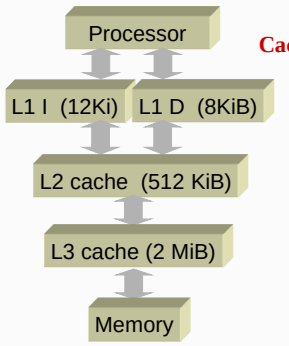
A Typical Memory Hierarchy
(With Two Levels of Cache)



17

Recent Typical Configurations





Pentium 4
Cache hierarchy (Gallatin)

Cycles: 2

Cycles: 19

Cycles: 43

Cycles: 206

19

Memory Hierarchy Basics

- When a word is not found in the higher level, a *miss* occurs:
  - Fetch word from lower level in hierarchy, requiring a higher latency reference
  - Also fetch the other words contained within the *block*
    - Takes advantage of spatial locality
  - Place block into cache in any location within its *set*, determined by address
    - block address MOD number of sets

Memory Hierarchy Operation

If an instruction or operand is required by the CPU, the levels of the memory hierarchy are searched for the item starting with the level closest to the CPU (Level 1 cache):

- **If the item is found, it's delivered to the CPU resulting in a cache hit.**
- **If the item is missing from an upper level, resulting in a miss, the level just below is searched.**
- **For systems with several levels of cache, the search continues with cache level 2, 3 etc.**
- **If all levels of cache report a miss then main memory is accessed.**
  - CPU ↔ cache ↔ memory: Managed by hardware.
- **If the item is not found in main memory resulting in a page fault, then disk (virtual memory), is accessed for the item.**
  - Memory ↔ disk: Managed by hardware and the operating system.

21

Memory Hierarchy Basics

- *n* sets => *n-way set associative*
  - *Direct-mapped cache* => one block per set
  - *Fully associative* => one set
- Writing to cache: two strategies
  - *Write-through*
    - Immediately update lower levels of hierarchy
  - *Write-back*
    - Only update lower levels of hierarchy when an updated block is replaced
  - Both strategies use *write buffer* to make writes asynchronous

Memory Hierarchy Basics

- Miss rate
  - Fraction of cache access that result in a miss
- Causes of misses
  - Compulsory
    - First reference to a block
  - Capacity
    - Blocks discarded and later retrieved
  - Conflict
    - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache

Memory Hierarchy Basics

$$\frac{\text{Misses}}{\text{Instruction}} = \frac{\text{Miss rate} \times \text{Memory accesses}}{\text{Instruction count}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

- Note that speculative and multithreaded processors may execute other instructions during a miss
  - Reduces performance impact of misses

Impact on Performance

- Suppose a processor executes at
  - Clock Rate = 200 MHz (5 ns per cycle)
  - CPI = 1.1
  - 50% arith/logic, 30% ld/st, 20% control
- Suppose that 10% of memory operations get 50 cycle miss penalty
- CPI = ideal CPI + average stalls per instruction
  = 1.1(cyc) +(0.30 (datamops/ins)
  x 0.10 (miss/datamop) x 50 (cycle/miss))
  = 1.1 cycle + 1.5 cycle = 2.6
- **58 % of the time the processor is stalled waiting for memory!**
- a 1% instruction miss rate would add an additional 0.5 cycles to the CPI!

25

Memory Hierarchy: Motivation
The Principle Of Locality

- Programs usually access a relatively small portion of their address space (instructions/data) at any instant of time (loops, data arrays).
- Two types of locality:
  - **Temporal Locality: If an item is referenced, it will tend to be referenced again soon.**
  - **Spatial locality: If an item is referenced, items whose addresses are close by will tend to be referenced soon .**

- The presence of locality in program behavior (e.g., loops, data arrays), makes it possible to satisfy a large percentage of the program's data accesses (both instructions and operands) using memory levels closer to the CPU.

26

Locality Example

Locality Example:

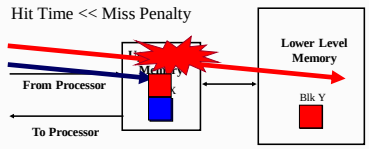
```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

- **Data**
  - Reference array elements in succession (stride-1 reference pattern): **Spatial locality**
  - Reference `sum` each iteration: **Temporal locality**
- **Instructions**
  - Reference instructions in sequence: **Spatial locality**
  - Cycle through loop repeatedly: **Temporal locality**

27

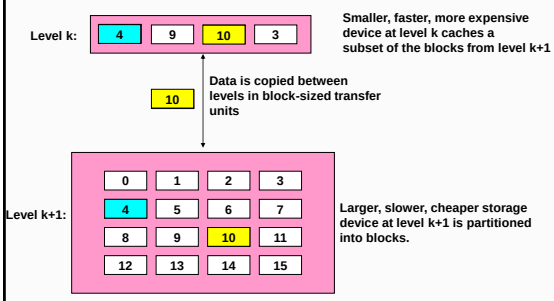
Memory Hierarchy: Terminology

- **A Block:** The smallest unit of information transferred between two levels.
- **Hit:** Item is found in a block in the upper level (example: Block X)
  - **Hit Rate: The fraction of memory accesses found in the upper level.**
  - **Hit Time: Time to access the upper level which consists of**
    memory access time + time to determine hit/miss
- Miss: Item needs to be retrieved from a block in the lower level (Block Y)
  - **Miss Rate = 1 - (Hit Rate)**
  - **Miss Penalty: Time to replace a block in the upper level +**
    Time to deliver the block to the processor (or the further-upper-level)
- Hit Time << Miss Penalty



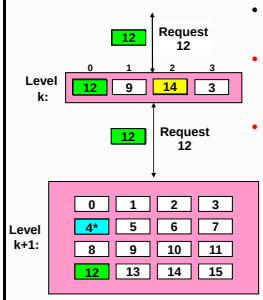
28

Caching in a Memory Hierarchy



29

General Caching Concepts



- Program needs object d, which is stored in some block b.
- Cache hit
  - Program finds b in the cache at level k. E.g., block 14.
- Cache miss
  - b is not at level k, so level k cache must fetch it from level k+1. E.g., block 12.
  - If level k cache is full, then some current block must be replaced (evicted). Which one is the "victim"?
    - Placement policy: where can the new block go? E.g., b mod 4
    - Replacement policy: which block should be evicted? E.g., LRU

30

Cache Design & Operation Issues

- Q1: Where can a block be placed in cache?
  (Block placement strategy & Cache organization)
  - **Fully Associative, Set Associative, Direct Mapped.**
- Q2: How is a block found if it is in cache?
  (Block identification)
  - **Tag/Block**
- Q3: Which block should be replaced on a miss?
  (Block replacement)
  - **Random, LRU.**
- Q4: What happens on a write?
  (Cache write policy)
  - **Write through, write back.**

31

Cache Organization & Placement Strategies

Placement strategies or mapping of a main memory data block onto cache block frame addresses divide cache into three organizations:

- **Direct mapped cache:** A block can be placed in one location only, given by:

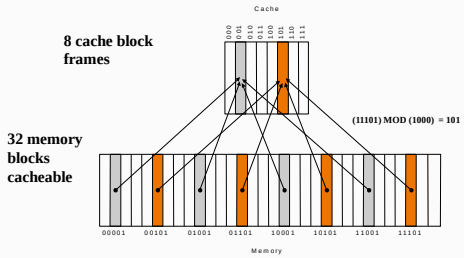
  (Block address) MOD (Number of blocks in cache)

  - Advantage: It is easy to locate blocks in the cache (only one possibility)
  - Disadvantage: Certain blocks cannot be simultaneously present in the cache (they can only have the same location)

32

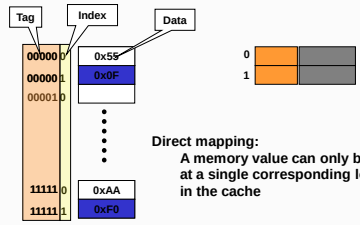
Cache Organization: Direct Mapped Cache

A block can be placed in one location only, given by:
(Block address) MOD (Number of blocks in cache)
In this case: (Block address) MOD (8)



33

Direct Mapping



Direct mapping:
A memory value can only be placed at a single corresponding location in the cache

34

Cache Organization & Placement Strategies

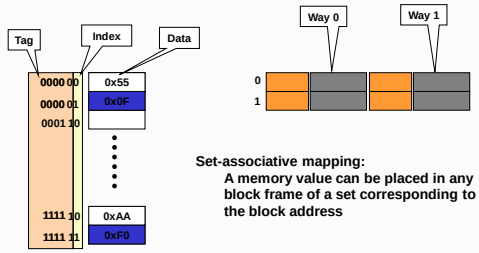
- **Fully associative cache:** A block can be placed anywhere in cache.
  - Advantage: No restriction on the placement of blocks. Any combination of blocks can be simultaneously present in the cache.
  - Disadvantage: Costly (hardware and time) to search for a block in the cache
- **Set associative cache:** A block can be placed in a restricted set of places, or cache block frames. A set is a group of block frames in the cache. A block is first mapped onto the set and then it can be placed anywhere within the set. The set in this case is chosen by:

  (Block address) MOD (Number of sets in cache)

  - If there are *n* blocks in a set the cache placement is called *n*-way set-associative, or n-associative.
  - A good compromise between direct mapped and fully associative caches (most processors use this method).

35

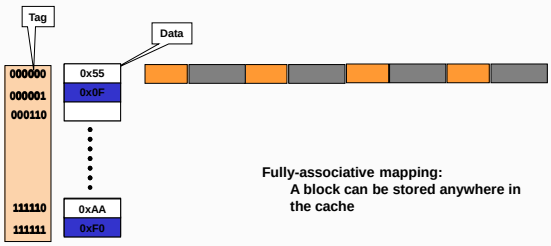
Set Associative Mapping (2-Way)



Set-associative mapping:
A memory value can be placed in any block frame of a set corresponding to the block address

36

Fully Associative Mapping



Fully-associative mapping:
A block can be stored anywhere in the cache

37

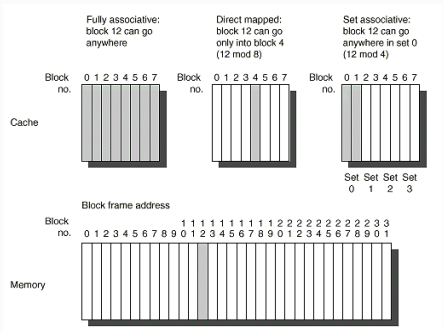
Types of Caches: Organization

Type of cache	Mapping of data from memory to cache	Complexity of searching the cache
Direct mapped (DM)		Easy search mechanism
Set-associative (SA)	A memory value can be placed in **any of a set of locations** in the cache	Slightly more involved search mechanism
Fully-associative (FA)	A memory value can be placed in **any location** in the cache	Extensive hardware resources required to search (CAM)

•DM and FA can be thought as special cases of SA
•DM → 1-way SA
•FA → All-way SA

38

Cache Organization Example



39

Cache Organization Tradeoff

- For a given cache size, there is a tradeoff between hit rate and complexity
- If L = number of lines (blocks) in the cache,
  L = Cache Size / Block Size

How many places for a block to go	Name of cache type	Number of Sets
1	Direct Mapped	L
n	n-way associative	L/n
L	Fully Associative	1

↑ Number of comparators needed to compare tags

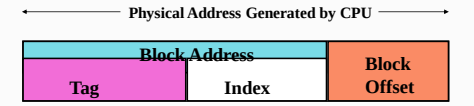
40

Locating A Data Block in Cache

- Each block in the cache has an address tag.
- The tags of every cache block that might contain the required data are checked in parallel.
- A valid bit is added to the tag to indicate whether this cache entry is valid or not.
- The address from the CPU to the cache is divided into:
  - **A block address, further divided into:**
    - An index field to choose a block set in the cache. (no index field when fully associative).
    - A tag field to search and match addresses in the selected set.
  - **A block offset to select the data from the block.**

41

Address Field Sizes



Block offset size = log2(block size)

Index size = log2(Total number of blocks/associativity)

Tag size = address size - index size - offset size

Number of Sets

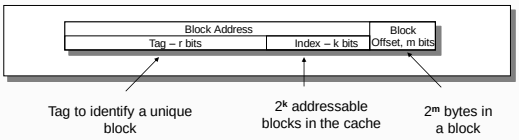
Mapping function:

Cache set or block frame number = Index =
= (Block Address) MOD (Number of Sets)

42

Locating A Data Block in Cache

- Increasing associativity shrinks index, expands tag
  - Block index not needed for fully associative cache



Direct-Mapped Cache Example

- Suppose we have a 16KB of data in a direct-mapped cache with 4 word blocks
- Determine the size of the tag, index and offset fields if we're using a 32-bit architecture
- Offset
  - need to specify correct byte within a block
  - block contains 4 words = 16 bytes = 2^4 bytes
  - need 4 bits to specify correct byte

Direct-Mapped Cache Example

- Index: (~index into an "array of blocks")
  - need to specify correct row in cache
  - cache contains 16 KB = 2^{14} bytes
  - block contains 2^4 bytes (4 words)

rows/cache =# blocks/cache (since there's one block/row)

= bytes/cache / bytes/row

= 2^{14} bytes/cache / 2^4 bytes/row

= 2^{10} rows/cache

need 10 bits to specify this many rows

Direct-Mapped Cache Example

- Tag: use remaining bits as tag
  - tag length = mem addr length
    - offset
    - index
    = 32 - 4 - 10 bits
    = 18 bits
  - so tag is leftmost 18 bits of memory address

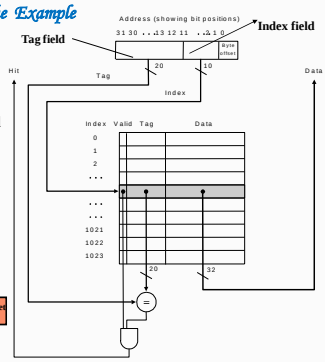
4KB Direct Mapped Cache Example

1K = 1024 Blocks
Each block = one 32b word

Cache for a 2^{32} bytes = 4 GB memory space

Mapping function:

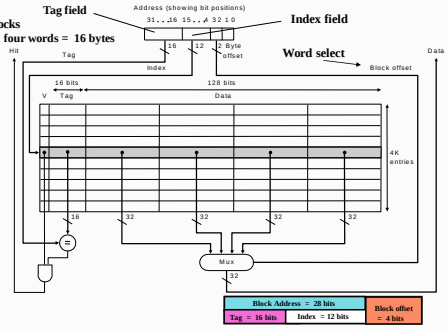
Cache Block frame number = (Block address) MOD (1024)



64KB Direct Mapped Cache Example

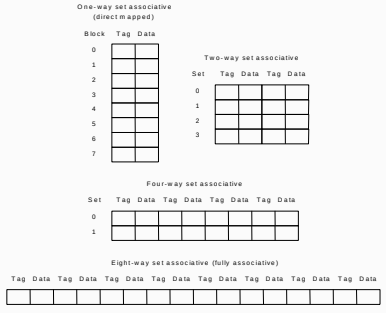
4K= 4096 blocks
Each block = four words = 16 bytes

Can cache up to 2^{32} bytes = 4 GB of memory



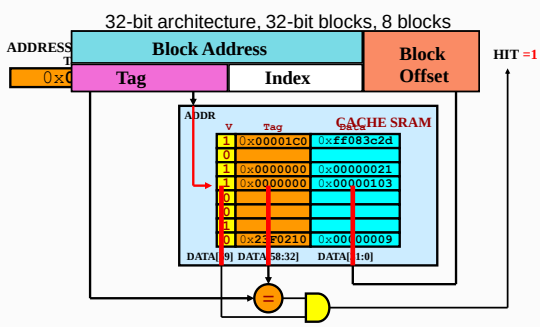
Mapping Function: Cache Block frame number = (Block address) MOD (4096)
Larger blocks take better advantage of spatial locality

Cache Organization: Set Associativity



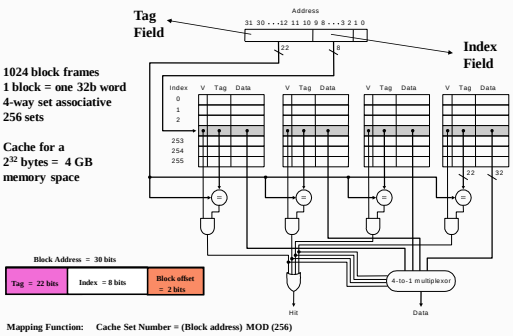
49

Direct-Mapped Cache Design



50

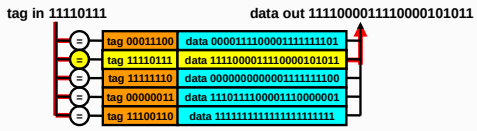
4KB Four-Way Set Associative Cache: MIPS Implementation Example



51

Fully Associative Cache Design

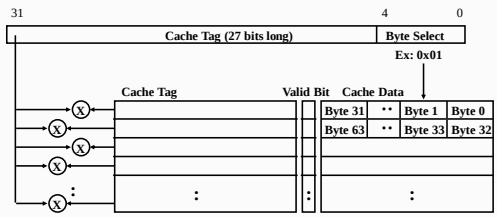
- Key idea: set size of one block
  - 1 comparator required for each block
  - No address decoding
  - Practical only for small caches due to hardware demands



52

Fully Associative

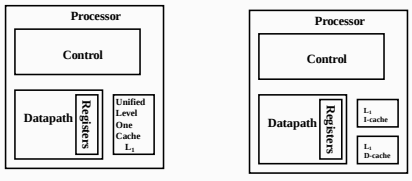
- Fully Associative Cache with N block frames
  - 0 bit for cache index
  - Compare the tags of all cache entries in parallel
  - Example: Block Size = 32 B blocks, we need N 27-bit comparators



53

Unified vs. Separate Level 1 Cache

- Unified Level 1 Cache
  A single level 1 cache is used for both instructions and data.
- Separate instruction/data Level 1 caches (Harvard Memory Architecture):
  The level 1 (L_1) cache is split into two caches, one for instructions (instruction cache, L_1 I-cache) and the other for data (data cache, L_1 D-cache).



54

Why have separate caches?

- Bandwidth: lets us access instructions and data in parallel (less structural hazards)
- Most programs don't modify their instructions
  - I-Cache can be simpler than D-Cache, since instruction references are never writes
- Instruction stream has high locality of reference, can get higher hit rates with small cache
  - Data references never interfere with instruction references

55

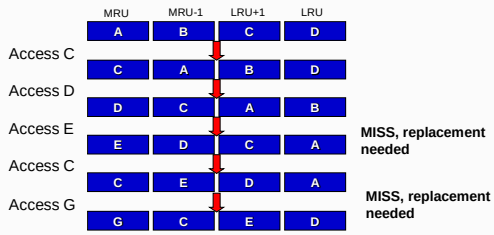
Cache Replacement Policy

When a cache miss occurs the cache controller may have to select a block of cache data to be removed from a cache block frame and replaced with the requested data, such a block is usually selected by one of two methods (for direct mapped cache, there is only one choice):

- **Random:**
  - Any block is randomly selected for replacement providing uniform allocation.
  - Simple to build in hardware.
  - The most widely used cache replacement strategy.
- **Least-recently used (LRU):**
  - Accesses to blocks are recorded and the block replaced is the one that was not used for the longest period of time.
  - LRU is *expensive* to implement, as the number of blocks to be tracked increases, and is usually approximated.

56

LRU Policy



57

Representative Miss Rates for Caches with Different Size, Associativity & Replacement Algorithm

Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64 KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

58

Cache and Memory Performance

Average Memory Access Time (AMAT), Memory Stall cycles

- ***The Average Memory Access Time*** **(AMAT)**: The average number of cycles required to complete a memory access request by the CPU.
- Memory stall cycles per memory access: The number of stall cycles added to CPU execution cycles for one memory access.
- For an ideal memory: AMAT = 1 cycle, this results in zero memory stall cycles.
- Memory stall cycles per memory access = AMAT -1
- Memory stall cycles per instruction =

  Memory stall cycles per memory access

  x Number of memory accesses per instruction

  = (AMAT -1) x (1 + fraction of loads/stores)

  Instruction Fetch

59

Cache Performance

Unified Memory Architecture

- For a CPU with a single level (L1) of cache for both instructions and data and no stalls for cache hits:

With ideal memory

Total CPU time = (CPU execution clock cycles + Memory stall clock cycles) x clock cycle time

Memory stall clock cycles =

(Reads x Read miss rate x Read miss penalty) +

(Writes x Write miss rate x Write miss penalty)

If write and read miss penalties are the same:

Memory stall clock cycles = Memory accesses x Miss rate x Miss penalty

60

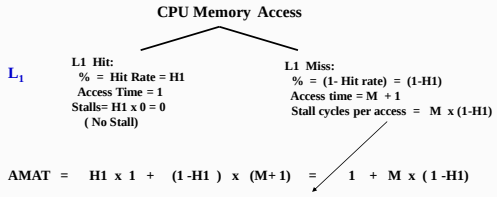
Cache Performance

Unified Memory Architecture

- CPUtime = Instruction count x CPI x Clock cycle time
- $CPI_{execution}$ = CPI with ideal memory
- CPI = $CPI_{execution}$ + MEM Stall cycles per instruction
- CPUtime = Instruction Count x ($CPI_{execution}$ + MEM Stall cycles per instruction) x Clock cycle time
- MEM Stall cycles per instruction = MEM accesses per instruction x Miss rate x Miss penalty
- CPUtime = IC x ($CPI_{execution}$ + MEM accesses per instruction x Miss rate x Miss penalty) x Clock cycle time
- Misses per instruction = Memory accesses per instruction x Miss rate
- CPUtime = IC x ($CPI_{execution}$ + Misses per instruction x Miss penalty) x Clock cycle time

61

Memory Access Tree For Unified Level 1 Cache



AMAT = H1 x 1 + (1 -H1) x (M+ 1) = 1 + M x (1 -H1)

Stall Cycles Per Access = AMAT - 1 = M x (1 -H1)

M = Miss Penalty
H1 = Level 1 Hit Rate
1- H1 = Level 1 Miss Rate

62

Cache Impact On Performance: An Example

Assuming the following execution and cache parameters:

- Cache miss penalty = 50 cycles
- Normal instruction execution CPI ignoring memory stalls = 2.0 cycles
- Miss rate = 2%
- Average memory references/instruction = 1.33

CPU time = IC x [$CPI_{execution}$ + Memory accesses/instruction x Miss rate x Miss penalty] x Clock cycle time

$CPUtime_{with\ cache}$ = IC x (2.0 + (1.33 x 2% x 50)) x clock cycle time
= IC x 3.33 x Clock cycle time

→ *Lower $CPI_{execution}$ increases the impact of cache miss clock cycles*

63

Cache Performance

Harvard Memory Architecture

For a CPU with separate or split level one (L1) caches for instructions and data (Harvard memory architecture) and no stalls for cache hits:

CPUtime = Instruction count x CPI x Clock cycle time

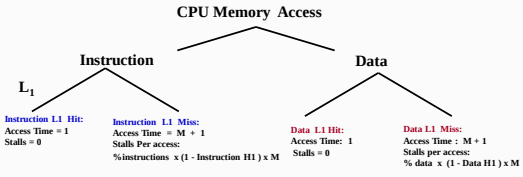
CPI = $CPI_{execution}$ + Mem Stall cycles per instruction

CPUtime = Instruction Count x ($CPI_{execution}$ + Mem Stall cycles per instruction) x Clock cycle time

Mem Stall cycles per instruction = Instruction Fetch Miss rate x Miss Penalty + Data Memory Accesses Per Instruction x Data Miss Rate x Miss Penalty

64

Memory Access Tree For Separate Level 1 Caches



Stall Cycles Per Access = % Instructions x (1 - Instruction H1) x M + % data x (1 - Data H1) x M

AMAT = 1 + Stall Cycles per access

65

Cache Write Strategies

66

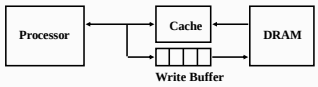
Cache Read/Write Operations

- Statistical data suggest that reads (*including instruction fetches*) dominate processor cache accesses (writes account for 25% of data cache traffic).
- In cache reads, a block is read at the same time while the tag is being compared with the block address (*searching*). If the read is a hit the data is passed to the CPU, if a miss it ignores it.
- In cache writes, modifying the block cannot begin until the tag is checked to see if the address is a hit.
- Thus for cache writes, tag checking cannot take place in parallel, and only the specific data requested by the CPU can be modified.
- Cache is classified according to the write and memory update strategy in place: write through, or write back.

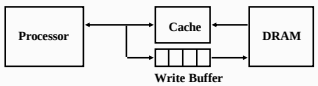
Cache Write Strategies

1 Write Though: Data is written to both the cache block and the main memory.

- The lower level always has fresh data; an important feature for I/O and multiprocessing.
- Easier to implement than write back.
- A write buffer is often used to reduce CPU write stall while data is written to memory.



Write Buffer for Write Through



- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO queue:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle

Cache Write Strategies

2 Write back: Data is written or updated only to the cache block.

- **Writes occur at the speed of cache**
- **The modified or dirty cache block is written to main memory later (e.g., when it's being replaced from cache)**
- **A status bit called a dirty bit, is used to indicate whether the block was modified while in cache; if not the block is not written to main memory.**
- **Uses less memory bandwidth than write through.**

Write misses

- If we try to write to an address that is not already contained in the cache; this is called a write miss.
- Let's say we want to store 21763 into Mem[1101 0110] but we find that address is not currently in the cache.

Index	V	Tag	Data
…			
110	1	00010	123456
…			

Address	Data
…	
1101 0110	6378
…	

- When we update Mem[1101 0110], should we *also* load it into the cache?

No write-allocate

- With a no-write allocate policy, the write operation goes directly to main memory *without* affecting the cache.

Mem[1101 0110] = 21763

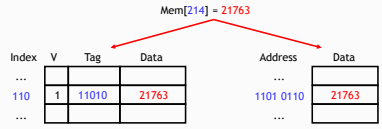
Index	V	Tag	Data
…			
110	1	00010	123456
…			

Address	Data
…	
1101 0110	21763
…	

- This is good when data is written but not immediately used again, in which case there's no point to load it into the cache yet.

Write Allocate

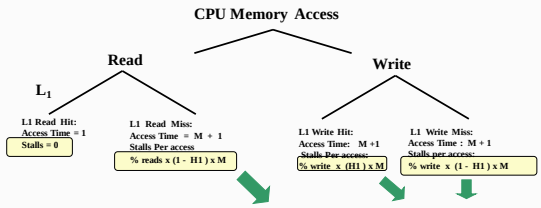
- A write allocate strategy would instead load the newly written data into the cache.



- If that data is needed again soon, it will be available in the cache.

73

Memory Access Tree, Unified L_1
Write Through, No Write Allocate, No Write Buffer

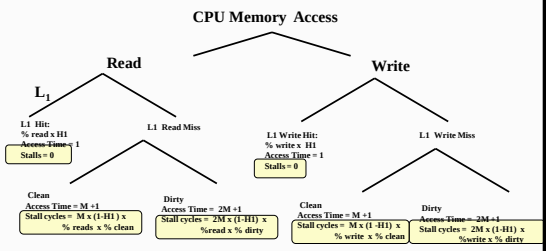


Stall Cycles Per Memory Access = % reads x (1 - H1) x M + % write x M

AMAT = 1 + % reads x (1 - H1) x M + % write x M

74

Memory Access Tree Unified L_1
Write Back, With Write Allocate

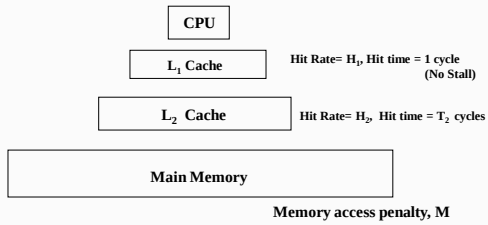


Stall Cycles Per Memory Access = (1-H1) x (M x % clean + 2M x % dirty)

AMAT = 1 + Stall Cycles Per Memory Access

75

2 Levels of Cache: L_1, L_2



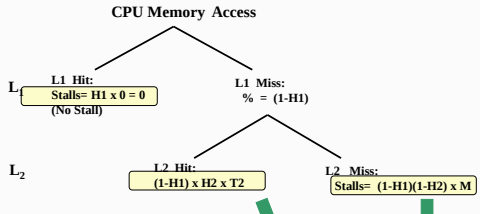
76

Miss Rates For Multi-Level Caches

- Local Miss Rate: This rate is the number of misses in a cache level divided by the number of memory accesses to this level. Local Hit Rate = 1 - Local Miss Rate
- Global Miss Rate: The number of misses in a cache level divided by the total number of memory accesses generated by the CPU.
- Since level 1 receives all CPU memory accesses, for level 1:
  - **Local Miss Rate = Global Miss Rate = 1 - H1**
- For level 2 since it only receives those accesses missed in level 1:
  - Local Miss Rate = Miss rate$_{L2}$ = 1- H2
  - Global Miss Rate = Miss rate$_{L1}$ x Miss rate$_{L2}$
    = (1- H1) x (1 - H2)

77

2-Level Cache Performance
Memory Access Tree



Stall cycles per memory access = (1-H1) x H2 x T2 + (1-H1)(1-H2) x M

AMAT = 1 + (1-H1) x H2 x T2 + (1-H1)(1-H2) x M

T2: L2 cache hit time in cycles

78

2-Level Cache Performance

CPUtime = IC x (CPI$_{execution}$ + Mem Stall cycles per instruction) x C

Mem Stall cycles per instruction = Mem accesses per instruction x Stall cycles per access

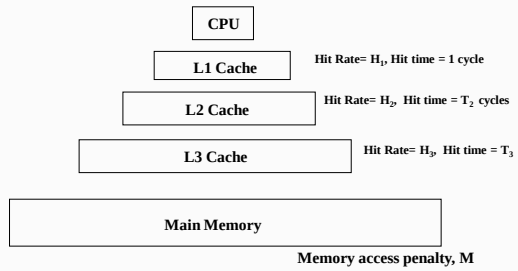
- For a system with 2 levels of cache, assuming no penalty when found in L_1 cache:
- Stall cycles per memory access =
  [miss rate L_1] x [Hit rate L_2 x Hit time L_2
  + Miss rate L_2 x Memory access penalty)] =
  (1-H1) x H2 x T2 + (1-H1)(1-H2) x M

L1 Miss, L2 Hit

L1 Miss, L2 Miss: Must Access Main Memory

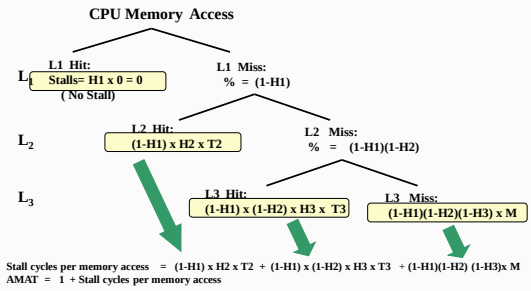
79

3 Levels of Cache



80

3-Level Cache Performance
Memory Access Tree
CPU Stall Cycles Per Memory Access



Stall cycles per memory access = (1-H1) x H2 x T2 + (1-H1) x (1-H2) x H3 x T3 + (1-H1)(1-H2) (1-H3)x M
AMAT = 1 + Stall cycles per memory access

81

3-Level Cache Performance

CPUtime = IC x (CPI$_{execution}$ + Mem Stall cycles per instruction) x C
Mem Stall cycles per instruction = Mem accesses per instruction x Stall cycles per access

For a system with 3 levels of cache, assuming no penalty when data found in L_1 cache:

Stall cycles per memory access =
[miss rate L_1] x [Hit rate L_2 x Hit time L_2
+ Miss rate L_2 x (Hit rate L3 x Hit time L_3
+ Miss rate L_3 x Memory access penalty)] =
(1-H1) x H2 x T2 + (1-H1) x (1-H2) x H3 x T3
+ (1-H1)(1-H2) (1-H3)x M

L1 Miss, L2 Hit

L2 Miss, L3 Hit

L1 Miss, L2 Miss: Must Access Main Memory

82

Reduce Miss Rate

83

83

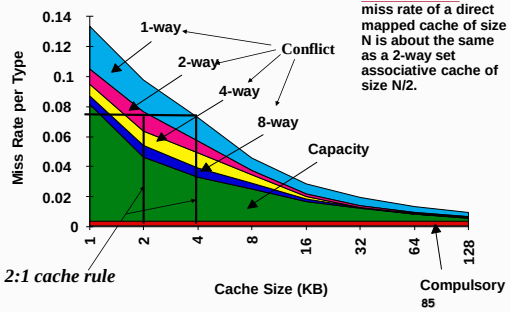
Reducing Misses (3 Cs)

- Classifying Misses: 3 Cs
  - ***Compulsory***—The first access to a block is not in the cache, so the block must be brought into the cache. These are also called *cold start misses* or *first reference misses*.
    (Misses even in infinite size cache)
  - ***Capacity***—If the cache cannot contain all the blocks needed during the execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.
    (Misses due to size of cache)
  - ***Conflict***—If the block-placement strategy is not fully associative, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called *collision misses* or *interference misses*.
    (Misses due to associativity and size of cache)

84

84

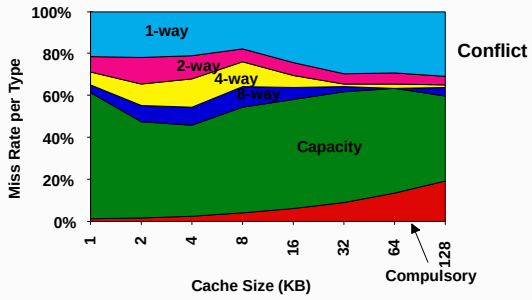
3Cs Absolute Miss Rates



85

85

3Cs Relative Miss Rate



86

86

How to Reduce the 3 Cs Cache Misses?

- Increase Block Size
- Increase Associativity
- Use a Victim Cache
- Use a Pseudo Associative Cache
- Hardware Prefetching

87

87

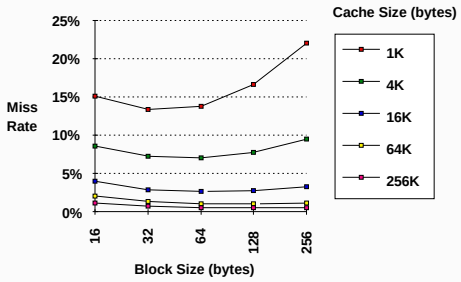
1. Increase Block Size

- One way to reduce the miss rate is to increase the block size
  - Take advantage of spatial locality
  - Reduce compulsory misses
- However, larger blocks have disadvantages
  - May increase the miss penalty (need to get more data)
  - May increase hit time
  - May increase conflict misses (smaller number of block frames)
- Increasing the block size can help, but don't overdo it.

88

88

1. Reduce Misses via Larger Block Size



89

89

2. Reduce Misses via Higher Associativity

- Increasing associativity helps reduce conflict misses (8-way should be good enough)
- 2:1 Cache Rule:
  - The miss rate of a direct mapped cache of size N is about equal to the miss rate of a 2-way set associative cache of size N/2
- Disadvantages of higher associativity
  - Need to do large number of comparisons
  - Need n-to-1 multiplexor for n-way set associative
  - Could increase hit time
    - Hit time for 2-way vs. 1-way external cache +10%, internal + 2%

90

90

Example: Avg. Memory Access Time vs. Associativity

Example: assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT=1 of direct mapped.

Cache Size (KB)	Associativity 1-way	2-way	4-way	8-way
1	7.65	6.60	6.22	5.44
2	5.90	4.90	4.62	4.09
4	4.60	3.95	3.57	3.19
8	3.30	3.00	2.87	2.59
16	2.45	2.20	2.12	2.04
32	2.00	1.80	1.77	1.79
64	1.70	1.60	1.57	1.59
128	1.50	1.45	1.42	1.44

(**Red** means memory access time not improved by higher associativity)

Does not take into account effect of slower clock on the rest of the program

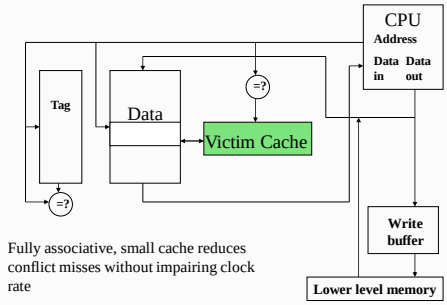
91 91

3. Reducing Misses via Victim Cache

- Add a small fully associative victim cache to hold data discarded from the regular cache
- When data not found in cache, check victim cache
- 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Get access time of direct mapped with reduced miss rate

92 92

3. Victim Cache



Fully associative, small cache reduces conflict misses without impairing clock rate

93 93

4. Reducing Misses via Pseudo-Associativity

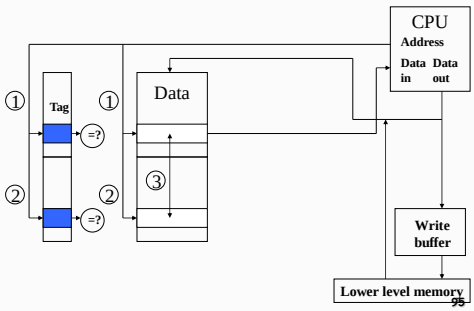
- How to combine fast hit time of direct mapped cache and the lower conflict misses of 2-way SA cache?
- Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit (slow hit).
- Usually check other half of cache by flipping the MSB of the index.

Drawbacks

- CPU pipeline is hard if hit takes 1 or 2 cycles
- Slightly more complex design

94 94

Pseudo Associative Cache



95 95

5. Hardware Prefetching

- Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in ***stream buffer***
  - On miss check stream buffer
- Works with data blocks too:
  - 1 data stream buffer gets 25% misses from 4KB DM cache; 4 streams get 43%
  - For scientific programs: 8 streams got 50% to 70% of misses from two 64KB, 4-way set associative caches (one for instructions and one for data)
- Prefetching relies on having extra memory bandwidth that can be used without penalty

96 96

Summary

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ \ accesses}{Instruction} \times \textbf{Miss rate} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ \ time$$

- 3 Cs: Compulsory, Capacity, Conflict Misses
- Reducing Miss Rate
  1. Larger Block Size
  2. Higher Associativity
  3. Victim Cache
  4. Pseudo-Associativity
  5. HW Prefetching Instr, Data

97 97

Pros and cons – Re-visit cache design choices

Larger cache block size

- Pros
  - Reduces miss rate
- Cons
  - Increases miss penalty

Important factors deciding cache performance: hit time, miss rate, miss penalty

98 98

Pros and cons – Re-visit cache design choices

Bigger cache

- Pros
  - Reduces miss rate
- Cons
  - May increases hit time
  - My increase cost and power consumption

99 99

Pros and cons – Re-visit cache design choices

Higher associativity

- Pros
  - Reduces miss rate
- Cons
  - Increases hit time

100 100

Pros and cons – Re-visit cache design choices

Multiple levels of caches

- Pros
  - Reduces miss penalty
- Cons
  - Increases cost and power consumption

101 101

Multilevel Cache Design Considerations

- Design considerations for L1 and L2 caches are very different
  - Primary cache should focus on minimizing hit time in support of a shorter clock cycle
    - Smaller cache with smaller block sizes
  - Secondary cache (s) should focus on reducing miss rate to reduce the penalty of long main memory access times
    - Larger cache with larger block sizes and/or higher associativity

102

Reducing Miss rate with programming

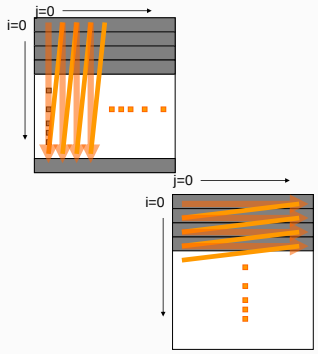
Examples:
cold cache, 4-byte words, 4-word cache blocks

```
int sumarraycols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Miss rate = ~ 100%

```
int sumarrayrows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Miss rate = ~1/N



Reducing Miss Penalty

104

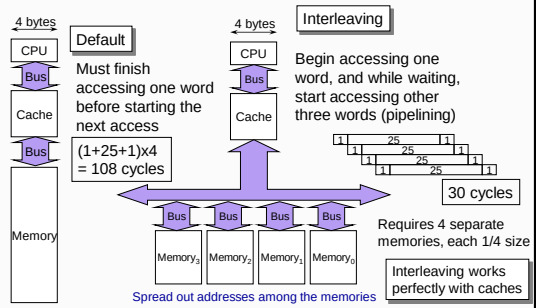
The cost of a cache miss

- For a memory access, assume:
  - 1 clock cycle to send address to memory
  - 25 clock cycles for each DRAM access (clock cycle 2ns, 50 ns access time)
  - 1 clock cycle to send each resulting data word
- Miss access time (4-word block)
  - 4 x (Address + access + sending data word)
  - 4 x (1 + 25 + 1) = 108
    = 108 cycles for each miss

This actually depends on the bus speed

105

Memory Interleaving



Sophisticated DRAMs provide support for this

106

Cache Optimization

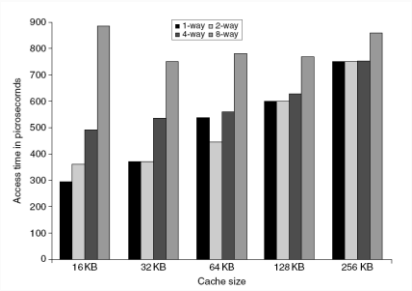
Six basic cache optimizations:

- Larger block size
  - Reduces compulsory misses
  - Increases capacity and conflict misses, increases miss penalty
- Larger total cache capacity to reduce miss rate
  - Increases hit time, increases power consumption
- Higher associativity
  - Reduces conflict misses
  - Increases hit time, increases power consumption
- Higher number of cache levels
  - Reduces overall memory access time
- Giving priority to read misses over writes
  - Reduces miss penalty
- Avoiding address translation in cache indexing
  - Reduces hit time

Ten Advanced Optimizations

1. Small and simple first level caches
   - Critical timing path:
     - addressing tag memory, then
     - comparing tags, then
     - selecting correct set
   - Direct-mapped caches can overlap tag compare and transmission of data
   - Lower associativity reduces power because fewer cache lines are accessed

L1 Size and Associativity



Access time vs. size and associativity

Way Prediction

2. To improve hit time, predict the way to pre-set mux
   - Mis-prediction gives longer hit time
   - Prediction accuracy
     - > 90% for two-way
     - > 80% for four-way
     - I-cache has better accuracy than D-cache
   - First used on MIPS R10000 in mid-90s
   - Used on ARM Cortex-A8

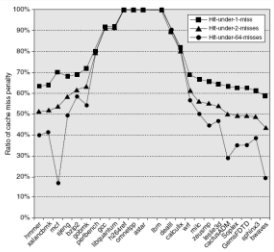
Extend to predict block as well
   - "Way selection"
   - Increases mis-prediction penalty

Pipelining Cache

3. Pipeline cache access to improve bandwidth
   - Examples:
     - Pentium: 1 cycle
     - Pentium Pro – Pentium III: 2 cycles
     - Pentium 4 – Core i7: 4 cycles
   - Increases branch mis-prediction penalty
   - Makes it easier to increase associativity

Nonblocking Caches

4. Allow hits before previous misses complete
   - "Hit under miss"
   - "Hit under multiple miss"
   - In general, processors can hide L1 miss penalty but not L2 miss penalty



Multibanked Caches

5. Organize cache as independent banks to support simultaneous access
   - ARM Cortex-A8 supports 1-4 banks for L2
   - Intel i7 supports 4 banks for L1 and 8 banks for L2
   - Interleave banks according to block address

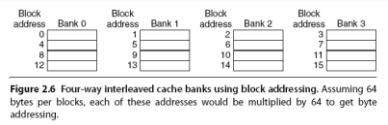


Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

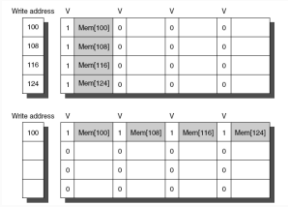
Critical Word First, Early Restart

6. Critical word first
   - Request missed word from memory first
   - Send it to the processor as soon as it arrives

Early restart
   - Request words in normal order
   - Send missed word to the processor as soon as it arrives
   - Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched

Merging Write Buffer

7. When storing to a block that is already pending in the write buffer, update write buffer (write merging)
   - Reduces stalls due to full write buffer
   - Do not apply to I/O addresses



No write merging

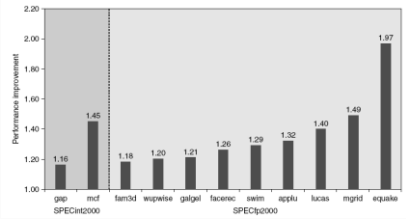
Write merging

Compiler Optimizations

8. Loop Interchange
   - Swap nested loops to access memory in sequential order
   - Blocking
     - Instead of accessing entire rows or columns, subdivide matrices into blocks
     - Improves locality if a block can fit in the cache

Hardware Prefetching

9. Fetch two blocks on miss (include next sequential block)



Pentium 4 Pre-fetching

Compiler Prefetching

10. Insert prefetch instructions before data is needed
    - Non-faulting: prefetch doesn't cause exceptions
    - Register prefetch
      - Loads data into register
    - Cache prefetch
      - Loads data into cache
    - Combine with loop unrolling and software pipelining

Summary

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined cache access	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	1	Used in L2 of both i7 and Cortex-A8
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware.
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs

Figure 2.11 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor, – means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.

Virtual Memory

- Originally invented to support program sizes larger than then-available physical memory
  - later on it finds applications in multi-programming and virtual machines
- Virtual memory is as large as the address space allowed by the ISA…but
  - only a portion of the address space resides in physical memory at any given time
  - the rest is kept on disks and brought into physical memory as needed

120

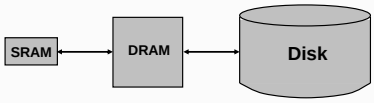
Motivations for Virtual Memory

- (1) Use Physical DRAM as a Cache for the Disk
  - Address space of a process (program) can exceed physical memory size
  - Sum of address spaces of multiple processes can exceed physical memory
- (2) Simplify Memory Management
  - Multiple processes reside in main memory.
    - Each process with its own address space
  - Only "active" code and data are actually in memory
    - Allocate more memory to process as needed.

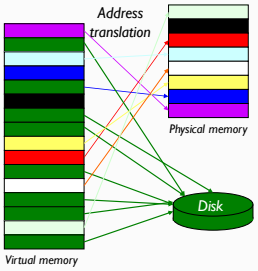
121

DRAM vs. SRAM as a "Cache"

- DRAM vs. disk is more extreme than SRAM vs. DRAM
  - Access latencies:
    - DRAM ~10X slower than SRAM
    - Disk ~**100,000X** slower than DRAM
  - Design decisions made for DRAM caches driven by enormous cost of misses



Virtual Memory



- Address translation
- Program "sees" entire VM address space
- Program is run in physical memory which is typically smaller than the address space
- Pages of address space are swapped in/out of disk storage as needed
- Strictly speaking VM is required to overcome limitation on the size of physical memory
  - but VM is extended in a natural way to support multiprogramming & memory protection
  - There are machines where the physical memory space is larger than the virtual memory space (e.g., PDP-11)

123

Advantages of Virtual Memory

- Abstraction of large and flat memory
  - program has a consistent view of a contiguous memory, even though physical memory is scrambled
  - Allows multiprogramming
  - relocation: allows the same program to run in any location in physical memory
- Protection
  - different processes are protected from each other
  - different pages can have different behavior (read-only; user/supervisor)
    - kernel code/data protected from user programs
- Sharing
  - can map same physical memory to multiple processes (shared memory)

124

How VM Works

- On program startup
  - OS loads part of the program into RAM; this includes enough code to start execution
  - if program size exceeds allocated RAM space the remainder is maintained on disk
- During execution
  - if program needs a code/data not resident in RAM, it fetches the segment from disk into RAM
  - if there is not enough room in RAM, some resident code/data is evicted from memory to make room
  - if evicted contents are "dirty", they are written to disk

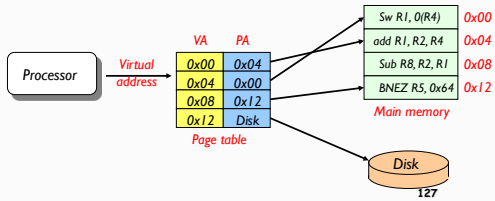
125

Address Translation

126

VA-to-PA Address Translation

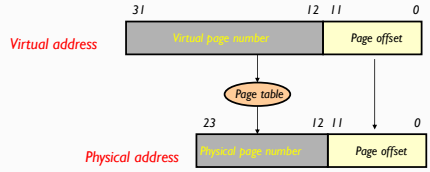
- Programs use virtual addresses (VA) for data & instructions
- VA translated to physical addresses (PA)
  - Usually organized in pages (paging) or segments
  - Paging systems use a "page table" for the translation
- Instruction/data fetched/updated in memory



127

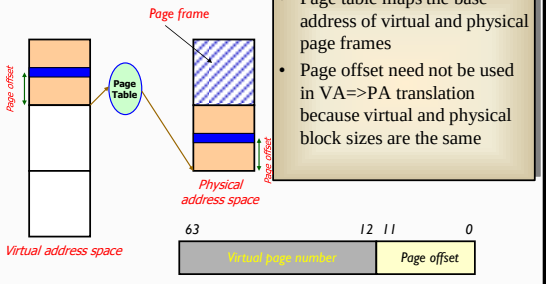
Page Table

- Memory organized in pages (similar to blocks in cache)
- Page size is usually 4-8 Kbytes



128

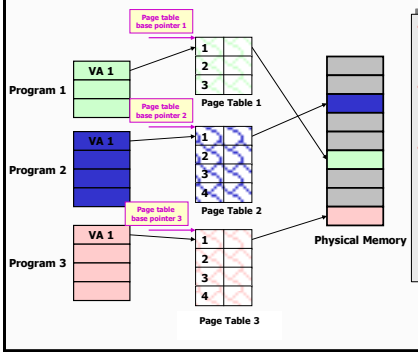
VA to PA Translation



- Page table maps the base address of virtual and physical page frames
- Page offset need not be used in VA=>PA translation because virtual and physical block sizes are the same

129

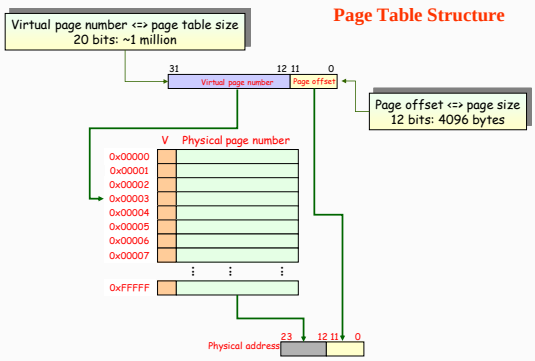
Multiprogramming View of VM



- Each application has a separate page table
- Page table contains an entry for each page
- Page tables are identified by the base pointers maintained by OS for individual processes

130

Page Table Structure



131

Determining Page Table Size

- Assume
  - 32-bit virtual address
  - 30-bit physical address
  - 4 KB pages => 12 bit page offset
  - Each page table entry is one word (4 bytes)
- How large is the page table?
  - Virtual page number = 32 - 12 = 20 bits
  - Number of entries = number of pages = 2^20
  - Total size = number of entries x bytes/entry
    = 2^20 x 4 = 4 Mbytes
  - Each process running needs its own page table

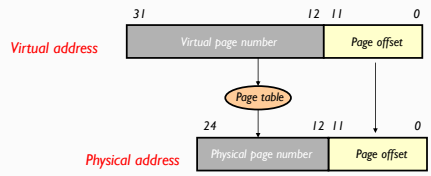
132

Page Fault

- How is it known whether the page is in memory?
  - Maintain a valid bit per page table entry
  - valid bit is set to INVALID if the page is not in memory
  - valid bit is set to VALID if the page is in memory
- Page fault occurs when a page is not in memory
  - fault results in OS fetching the page from disk into DRAM
  - if DRAM is full, OS must evict a page (victim) to make room
  - if victim is dirty OS updates the page on disk before fetch
  - OS changes page table to reflect turnover
- After a page fault and page-fetching, execution resumes at the instruction which caused the fault

Accelerating Virtual Memory Operations

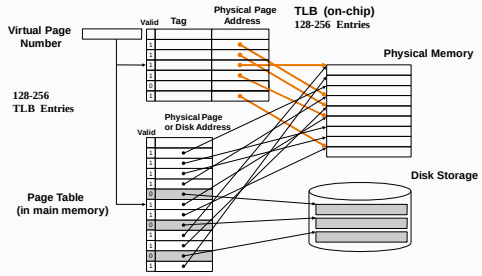
VM => PM Translation



- Each program memory reference requires two memory accesses: one for VM => PM mapping and one for actual data/instruction
  - must make page table lookup as fast as possible
- Page table too big to keep in fast memory (SRAM) in its entirety
  - store page table in main memory
  - cache a portion of the page table in TLB (Translation Look-Aside Buffer)

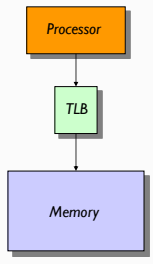
Speeding Up Address Translation: Translation Lookaside Buffer (TLB)

- TLB: A small and fast on-chip memory structure used for address translations.
- If a virtual address is found in TLB (a TLB hit), the page table in main memory is not accessed.

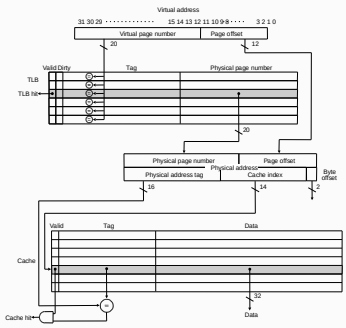


Translation Look-Aside Table (TLB)

- TLB maintains a list of most-recently used pages
- Similar to instruction & data cache
  - virtual address is used to index into the TLB
  - TLB entry contains physical address
  - takes ~ 1 clock cycle
- What if VM=>PM lookup and data/instruction lookup takes more than 1 clock cycle?
  - To avoid TLB lookup, remember the last VM => PM translation
  - if same page is referenced, TLB lookup can be avoided



TLB / Cache Interaction

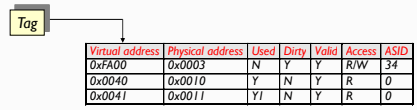


TLB and Context Switch

- In multi-programming we need to use TLB for the active process
  - What to do with TLB at context switch?
- Too costly to clear TLB on every context switch
- Keep track of PTE in TLB per process using ASID

139

TLB Organization



Virtual address	Physical address	Used	Dirty	Valid	Access	ASID
0xFA00	0x0003	N	Y	Y	R/W	34
0x0040	0x0010	Y	N	Y	R	0
0x0041	0x0011	Y	N	Y	R	0

Additional info per page
- Dirty: page modified (if the page is swapped out, should the disk copy be updated?)
- Valid: TLB entry valid
- Used: Recently used or not (for selecting a page for eviction)
- Access: Read/Write
- ASID:Address Space ID

140

Associativity of VM

- Cache miss penalty: 8-150 clock cycles
- VM miss penalty: 1,000,000 - 10,000,000 clock cycles
- Because of the high miss penalty, VM design minimizes miss rate by allowing full associativity for page placement

141

Write Strategies

- Disk I/O slow (millions of clock cycles)
- Always write-back; never write-through
- Use dirty bit to decide whether to write disk before eviction
- Smart disk controllers buffers writes
  - copy replaced page in buffer
  - read new page into main memory
  - write from buffer to disk

142

Virtual Machines

- Supports isolation and security
- Sharing a computer among many unrelated users
- Enabled by raw speed of processors, making the overhead more acceptable

- Allows different ISAs and operating systems to be presented to user programs
  - "System Virtual Machines"
  - SVM software is called "virtual machine monitor" or "hypervisor"
  - Individual virtual machines run under the monitor are called "guest VMs" or "guest OSes"

I/O Performance Measures

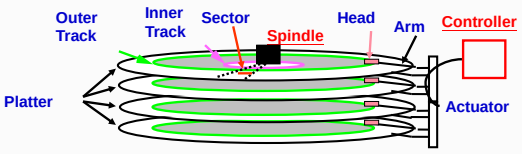
- I/O bandwidth (throughput) – amount of information that can be input (output) and communicated across an interconnect (e.g., a bus) to the processor/memory (I/O device) per unit time
  1. How much data can we move through the system in a certain time?
  2. How many I/O operations can we do per unit time?

- I/O response time (latency) – the total elapsed time to accomplish an input or output operation
  - An especially important performance metric in real-time systems

- Many applications require both high throughput and short response times

144

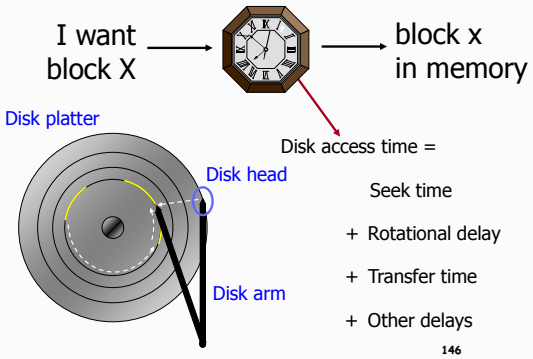
Hard Disk Performance



- Disk Latency = Seek Time + Rotation Time + Transfer Time + Controller Overhead
- Seek Time depends on the moving distance and moving speed of arms
- Rotation Time depends on how fast the disk rotates and how far sector is from head
- Transfer Time depends on data rate (bandwidth) of disk and size of request

145

Disk Access Time



146

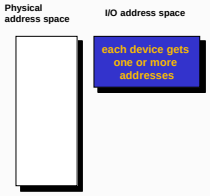
Communication of I/O Devices and Processor

How does the processor command the I/O devices?

– Special I/O instructions
  - Must specify both the device (port number) and the command

– For example:

inp reg, port ; register←port
out port, reg ; port→register

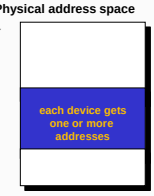


147

Communication of I/O Devices and Processor

How does the processor command the I/O devices?

– Memory-mapped I/O
  - Portions of the memory address space are assigned to I/O devices
  - Read and writes to those memory addresses are interpreted as commands to the I/O devices
  - Load/stores to the I/O address space can only be done by the OS



Communication of I/O Devices and Processor

How does the I/O device communicate with the processor?

– Polling – the processor periodically checks the status of an I/O device to determine its need for service
  - Processor is totally in control – it also does all the work
  - Can waste a lot of processor time due to speed differences

– Interrupt-driven I/O – the I/O device issues an interrupt to the processor to indicate that it needs attention

149

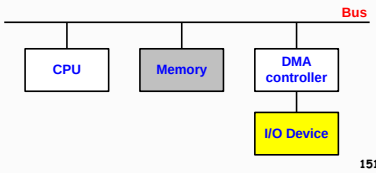
Direct-Memory Access (DMA)

- Interrupt-driven IO relieves the CPU from waiting for every IO event
- But the CPU can still be bogged down if it is used in transferring IO data.
  – Typically blocks of bytes.
- For high-bandwidth devices (like disks) interrupt-driven I/O would consume a *lot* of processor cycles

150

DMA

- DMA – the I/O controller has the ability to transfer data directly to/from the memory without involving the processor



151

I/O Buses

- Connect I/O devices (channels) to memory.
  - Many types of devices are connected to a bus.
  - Have a wide range of bandwidth requirements for the devices connected to a bus.
  - Typically follow a bus standard, e.g., PCI, SCSI.
- Clocking schemes:
  - Synchronous: The bus includes a clock signal in the control lines and a fixed protocol for address and data relative to the clock
  - Asynchronous: The bus is self-timed and uses a handshaking protocol between the sender and receiver

152

RAID (Redundant Array of Inexpensive Disks)

RAID

- To increase the availability and the performance (bandwidth) of a storage system, instead of a single disk, a set of disks (disk arrays) can be used.
- Similar to memory interleaving, data can be spread among multiple disks (*striping*), allowing simultaneous access to the data, improving the throughput and latency besides availability.
- However, the reliability of the system drops (*n* devices have 1/*n* the reliability of a single device).

Dependability Measures

- Reliability: mean time to failure (MTTF)
- Service interruption: mean time to repair (MTTR)
- Mean time between failures
  - MTBF = MTTF + MTTR
- Availability = MTTF / (MTTF + MTTR)
- Improving Availability
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: improved tools and processes for diagnosis and repair

Array Reliability

- **Reliability of N disks = Reliability of 1 Disk ÷N**

  50,000 Hours ÷ 70 disks = 700 hours

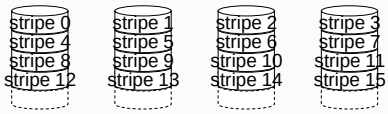
  Disk system Mean Time To Failure (MTTF): Drops from 6 years to 1 month!

Disks without redundancy are too unreliable to be useful!

RAID

- A disk array's availability can be improved by adding redundant disks:
  - If a single disk in the array fails, the lost information can be reconstructed from redundant information.
- This leads to a technology known as RAID - Redundant Array of Inexpensive Disks.
  - Depending on the number of redundant disks and the redundancy scheme used, RAIDs are classified into levels.
  - At least 6 levels of RAID (0-5) are accepted by the industry.
  - Level 2 is not commercially available

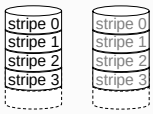
RAID-0



- Striped, non-redundant
  - Parallel access to multiple disks
  - ➔Excellent data transfer rate
  - ➔Excellent I/O request processing rate (for large stripes) if the controller supports independent Reads/Writes
  - ➔Not fault tolerant (**AID**)
- Typically used for applications requiring high performance for non-critical data (e.g., video streaming and editing)

RAID-1 - Mirroring

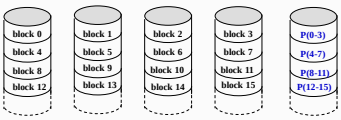
- Called mirroring or shadowing, uses an extra disk (mirror) for each disk in the array
  - costly form of redundancy (but some FS, e.g., GFS, makes 3 copies)
- Whenever data is written to one disk, the data is also written to the mirror: good for reads (lower latency), fair for writes
- If a disk fails, the system goes to the mirror and gets the desired data.
- Fast, but very expensive.
- Typically used in system drives and critical files
  - Banking, insurance data
  - e-commerce servers



RAID-4 - Block-interleaved Parity

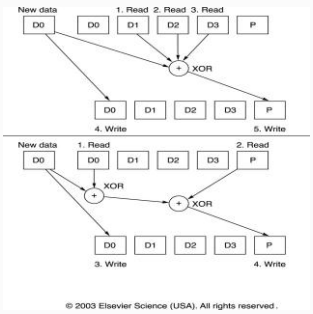
- In RAID 3, every read or write needs to go to all disks since bits are interleaved among the disks.
- Performance of RAID 3:
  - Only one request can be serviced at a time
  - Poor I/O request rate
  - Excellent data transfer rate
  - Typically used in large I/O request size applications, such as imaging or CAD
- RAID 4: If we distribute the information block-interleaved, where a disk sector is a block, then for normal reads different reads can access different segments in parallel. Only if a disk fails will we need to access all the disks to recover the data.

RAID-4: Block Interleaved Parity



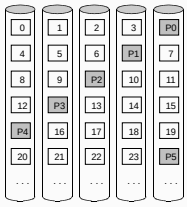
- Allow for parallel access by multiple I/O requests
- Doing multiple small reads is now faster than before.
- A write, however, is a different story since we need to update the parity information for the block.
- Large writes (full stripe), update the parity:
  P' = d0' ⊕ d1' ⊕ d2' ⊕ d3';
- Small writes (eg. write on d0), update the parity:
  P = d0 ⊕ d1 ⊕ d2 ⊕ d3
  P' = d0' ⊕ d1 ⊕ d2 ⊕ d3 = d0' ⊕ d0 ⊕ P;
- However, writes are still very slow since parity disk is the bottleneck.

RAID-4: Small Writes



RAID-5 - Block-interleaved Distributed Parity

RAID 5 distributes the parity blocks among all the disks.



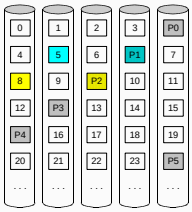
RAID 5

Why is this helpful?

RAID-5 - Block-interleaved Distributed Parity

This allows *some* writes to proceed in parallel

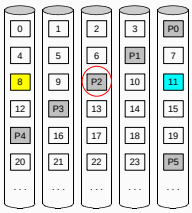
- For example, writes to blocks 8 and 5 can occur simultaneously.



RAID 5

RAID-5 - Block-interleaved Distributed Parity

- However, writes to blocks 8 and 11 cannot proceed in parallel.



RAID 5

Performance of RAID-5 - Block-interleaved Distributed Parity

- **Performance of RAID-5**
  - I/O request rate: excellent for reads, good for writes
  - Data transfer rate: good for reads, good for writes
  - Typically used for high request rate, read-intensive data lookup
  - **File and Application servers, Database servers, WWW, E-mail, and News servers, Intranet servers**
- Widely used.

RAID-6 – Row-Diagonal Parity

- To handle 2 disk errors
  - In practice, another disk error can occur before the first problem disk is repaired
- Use p-1 data disks, 1 row-parity disk, 1 diagonal-parity disk
- If any two of the p+1 disks fail, data can still be recovered

Data Disk 0	Data Disk 1	Dada Disk 2	Dada Disk 3	Row Parity Disk	Diagonal Parity Disk
0	1	2	3	4	0
1	2	3	4	0	1
2	3	4	0	1	2
3	4	0	1	2	3

Dependability

Definitions

- Examples on why precise definitions so important for reliability
- **Is a programming mistake a fault, error, or failure?**
  - **Are we talking about the time it was designed or the time the program is run?**
  - **If the running program doesn't exercise the mistake, is it still a fault/error/failure?**
- **If an alpha particle hits a DRAM memory cell, is it a fault/error/failure if it doesn't change the value?**
  - **Is it a fault/error/failure if the memory doesn't access the changed bit?**
  - **Did a fault/error/failure still occur if the memory had error correction and delivered the corrected value to the CPU?**

IFIP Standard terminology

- **Computer system *dependability*: quality of delivered service such that reliance can be justifiably placed on the service**
- ***Service* is observed *actual behavior* as perceived by other system(s) interacting with this system's users**
- **Each module has ideal *specified behavior*, where *service specification* is agreed description of expected behavior**
- **A system *failure* occurs when the actual behavior deviates from the specified behavior**
- **failure occurred because an *error*, a defect in module**
- **The cause of an error is a *fault***
- **When a fault occurs it creates a *latent error*, which becomes *effective* when it is activated**
- **When error actually affects the delivered service, a failure occurs (time from error to failure is *error latency*)**

Why multi-core ?

- Difficult to make single-core clock frequencies even higher
- Deeply pipelined circuits:
  - Heat problems
  - Clock problems
  - Efficiency (Stall) problems
- Doubling issue rates above today's 3-6 instructions per clock, say to 6 to 12 instructions, is extremely difficult
  - issue 3 or 4 data memory accesses per cycle,
  - rename and access more than 20 registers per cycle, and
  - fetch 12 to 24 instructions per cycle.
- Many new applications are multithreaded

A general trend in computer architecture is to shift towards more parallelism through more processors or processor cores

171

Thread-Level Parallelism (TLP)

- This is parallelism on a more coarse scale
- Server can serve each client in a separate thread (Web server, database server)
- A computer game can do AI, graphics, and sound in three separate threads
- Single-core superscalar processors cannot fully exploit TLP
- Multi-core architectures are the next step in processor evolution: explicitly exploiting TLP

172

Thread-Level Parallelism

173

TLP

- 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

How to exploit TLP?

- Execute instructions from multiple threads on a single processor
  - Coarse-grain, fine-grain, SMT (Simultaneous Multi-Threading)
- Execute multiple threads on multiple processors
  - "Anything that can be threaded today will map efficiently to multi-core"

175

SMT – Simultaneous Multi-Threading

A variation on multithreading that uses the resources of a multiple-issue, dynamically scheduled processor (superscalar) to exploit both program ILP and thread-level parallelism (TLP)

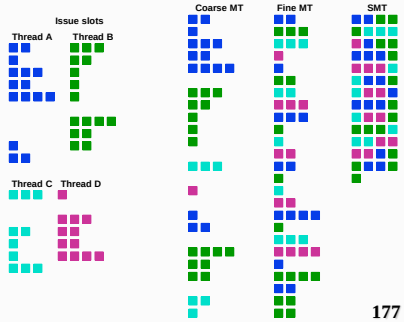
With register renaming and dynamic scheduling, multiple instructions from independent threads can be issued in one cycle without regard to dependencies among them

Need separate rename tables (ROBs) for each thread

Need the capability to commit from multiple threads (i.e., from multiple ROBs) in one cycle

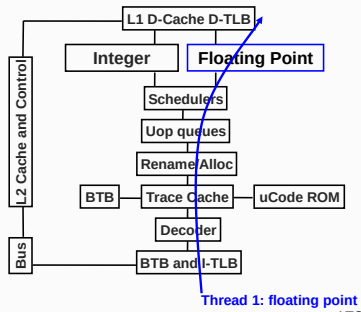
176

TLP a 4-issue superscalar processor



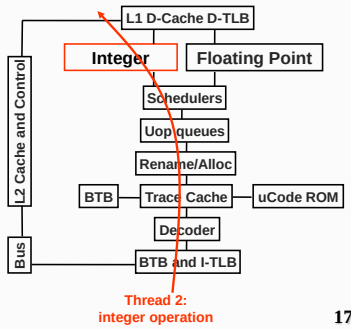
177

Without SMT, only a single thread can run at any given time



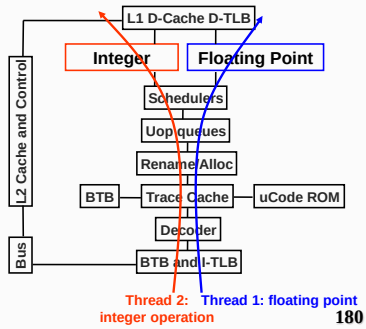
178

Without SMT, only a single thread can run at any given time



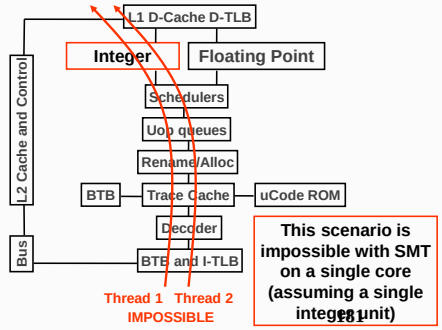
179

SMT processor: both threads can run concurrently



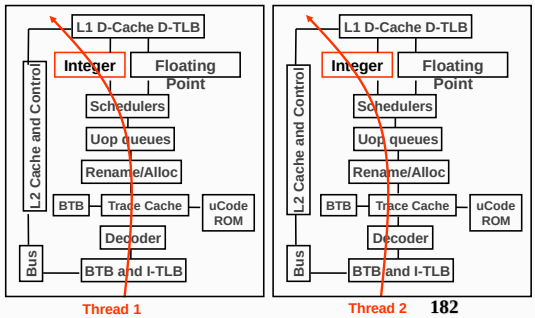
180

But: Can't simultaneously use the same functional unit



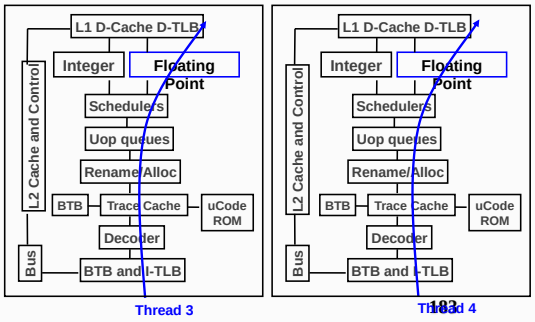
181

Multi-core:
threads can run on separate cores



182

Multi-core:
threads can run on separate cores



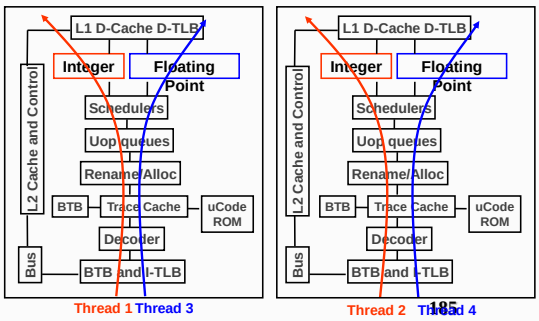
183

Combining Multi-core and SMT

- Cores can be SMT-enabled (or not)
- The different combinations:
  - Single-core, non-SMT: standard uniprocessor
  - Single-core, with SMT
  - Multi-core, non-SMT
  - Multi-core, with SMT
- The number of SMT threads:
  2, 4, or sometimes 8 simultaneous threads
  - Intel calls it "hyper-threading"

184

SMT Dual-core: all four threads can run concurrently



185

High-Performance Computing

186

When Do We Need High Performance Computing?

Case1

–To do a time-consuming operation in **less time**

- I am an aircraft engineer
- I need to run a simulation to test the stability of the wings at high speed
- I'd rather have the result in 5 minutes than in 5 days so that I can complete the aircraft final design sooner.

187

When Do We Need High Performance Computing?

Case 2

– To do a **high number of operations** per seconds

- I am an engineer of Amazon.com
- My Web server gets 10,000 hits per seconds
- I'd like my Web server and my databases to handle 10,000 transactions per seconds so that customers do not experience bad delays

188

Multiprocessing

- Multiprocessing (Parallel Processing): *Concurrent* execution of tasks (programs) using *multiple computing, memory and interconnection resources.*

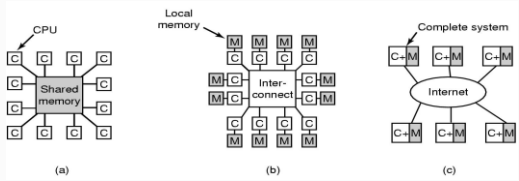
  Use multiple resources to solve problems faster

- Provides an alternative to faster clock for performance
  - *Assuming a doubling of effective processor performance every 2 years, 1024-Processor system (assuming linear performance gain) can get you the performance that it would take 20 years for a single-processor system to deliver*
- Using multiple processors to solve a single problem
  - Divide problem into many small pieces
  - Distribute these small problems to be solved by multiple processors simultaneously

189

Multiprocessing

- For the last 30+ years multiprocessing has been seen as the best way to produce orders of magnitude performance gains.
  - Double the number of processors, get (theoretically) double performance (less than 2 times the cost).
- It turns out that the ability to develop and deliver software for multiprocessing systems induces impediment to wide adoption.



190

Performance Potential: Another View

- Gustafson view
  - Parallel portion increases as the problem size increases
    - Serial time fixed (at s)
    - Parallel time proportional to problem size (true most of the time)
    - Gustafson's Law: Speedup(N) = N - β(N-1)
      - N: number of processors, β : weight of non-parallelizable part

```
Old Serial:    SSPPPPPP
6 processors:  SSPPPPPP
                 PPPPPP
                 PPPPPP
                 PPPPPP
                 PPPPPP
                 PPPPPP
Hypothetical Serial:
 SSPPPPPP PPPPPP PPPPPP PPPPPP PPPPPP PPPPPP
```

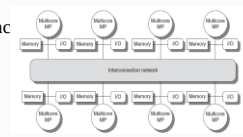
  - Speedup(6) = (8+5*6)/8 = 4.75
    - β =? in this calculation?
  - Speedup(N) = N(1- β) + β; Speedup'(∞) → ∞!!!!

This means we can achieve higher speedup by processing larger problems.

191

A Few Types

- 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/latenc (NUMA)
  - Processors connected via direct (switched) and non-direct (multi-hop) interconnection networks

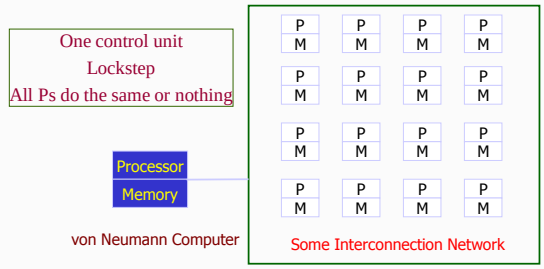


Flynn's Taxonomy of Computing

- *SISD* (Single Instruction, Single Data):
  - Typical uniprocessor systems that we've studied throughout this course.
  - Uniprocessor systems can time share and still be SISD.
- *SIMD* (Single Instruction, Multiple Data):
  - Multiple processors *simultaneously* executing the *same instruction* on different data.
  - Specialized applications (e.g., image processing).
- *MIMD* (Multiple Instruction, Multiple Data):
  - Multiple processors *autonomously* executing *different instructions on different data*.
  - Keep in mind that the processors are working together to solve a single problem.
  - This is a more general form of multiprocessing, and can be used in numerous applications

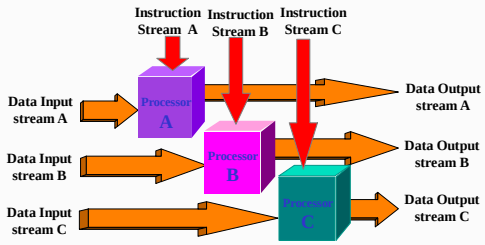
193

SIMD Systems



194

MIMD Architecture

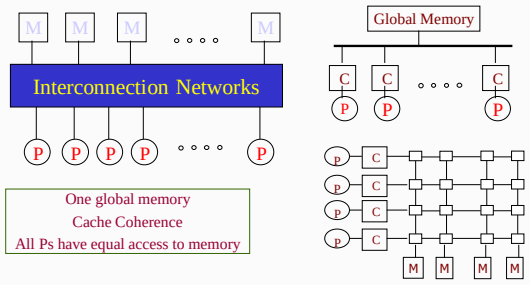


Unlike SIMD, MIMD computer works asynchronously.

- Shared memory (tightly coupled) MIMD
- Distributed memory (loosely coupled) MIMD

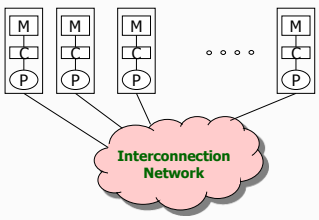
195

MIMD Shared Memory Systems



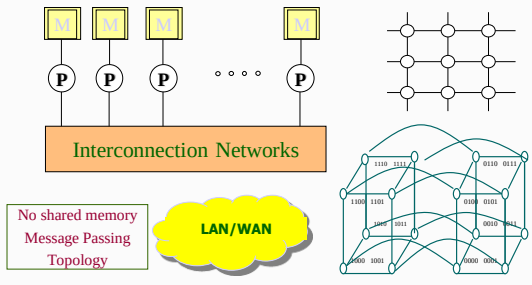
196

Cache Coherent NUMA



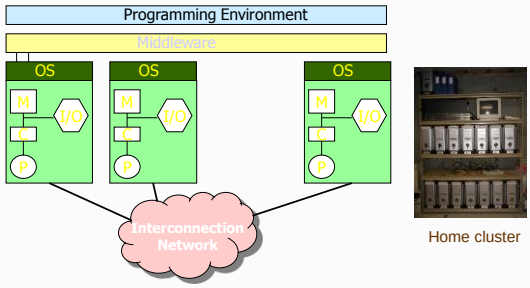
197

MIMD Distributed Memory Systems



198

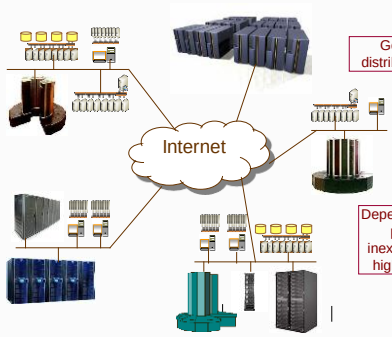
Cluster Architecture



Home cluster

199

Grids



Geographically distributed platforms.

Dependable, consistent, pervasive, and inexpensive access to high end computing.

200

SIMD

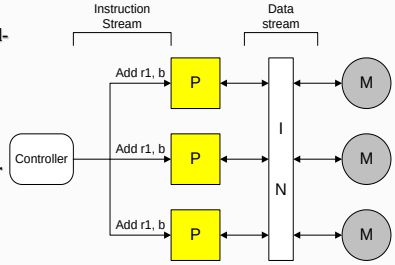
201

SIMD Parallel Computing

It can be a stand-alone multiprocessor

Or

Embedded in a single processor for specific applications (MMX)



SIMD execution

202

SIMD Multiprocessing

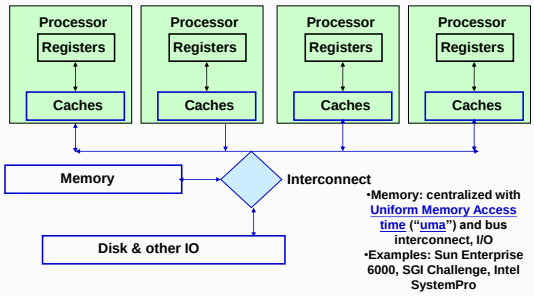
- Traditional vector computers are typical SIMD systems
- In the late 80s and early 90s, many SIMD machines were commercially available (e.g., Connection machine has 64K ALUs, and MasPar has 16K ALUs)
- GPU revives the SIMD computation, and is widely used in high-performance computers
- SPMD—Single Program, Multiple Data

203

Cache Coherence

204

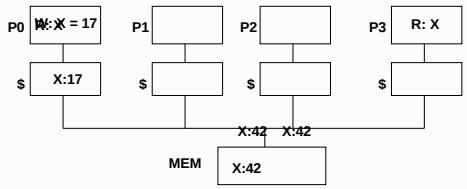
Shared Memory Multiprocessor



- Memory: centralized with Uniform Memory Access time ("uma") and bus interconnect, I/O
- Examples: Sun Enterprise 6000, SGI Challenge, Intel SystemPro

205

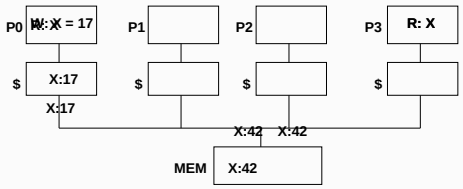
Cache Coherence Problem



- Processor 3 does not see the value written by processor 0

206

Write Through does not help



- Processor 3 sees 42 in cache (does not get the correct value (17) from memory.

207

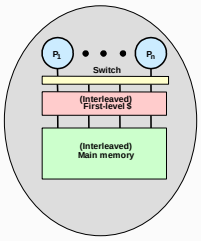
Why Don't Processors Share Cache

Advantages

- Cache placement identical to single cache
  - only one copy of any cached block

Disadvantages

- Bandwidth limitation



208

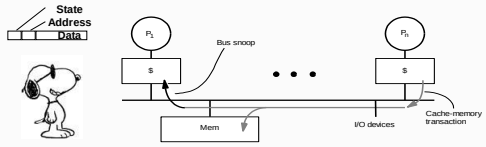
Cache Coherence

- Coherence
  - All reads by any processor 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

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

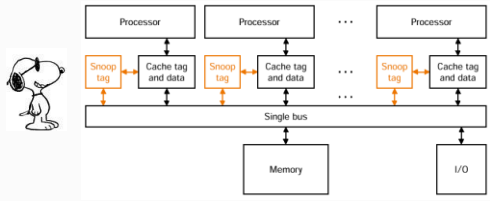
Distributed Cache: Snoopy Cache-Coherence Protocols



- Bus is a broadcast medium & caches know what they have
  - bus protocol: arbitration, command/addr, data
  - => Every device observes every transaction

211

Snooping Cache Coherency

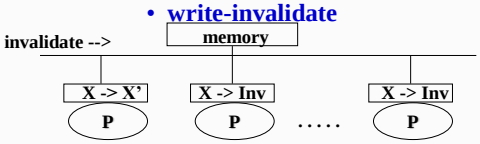


- **Cache Controller "snoops" all transactions on the shared bus**
  - **A transaction is a relevant transaction if it involves a cache block currently contained in this cache**
    - **take action to ensure coherence (invalidate, update, or supply value)**

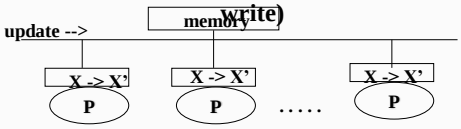
212

Hardware Cache Coherence

- **write-invalidate**



- **write-update (also called distributed write)**



An Example Snoopy Protocol

- Invalidation protocol, write-back cache
  - "invalidate" request on memory bus
- Each cache block is in one state (track these):
  - Shared : multiple caches potentially have copies of the block; the block can be read
  - OR Modified : this cache has the only copy of the block; the block is writeable and dirty
  - OR Invalid : the block contains no valid data

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

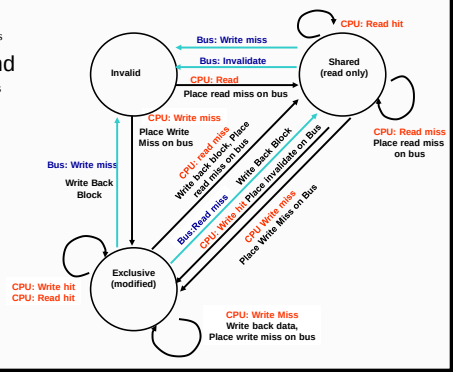
- Write update
  - On write, update all copies

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, since 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 share data: place cache block on bus 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.

Snoopy-Cache State Machine

- State machine for CPU requests for each cache block and for bus requests for each cache block



Performance

Coherence influences cache miss rate

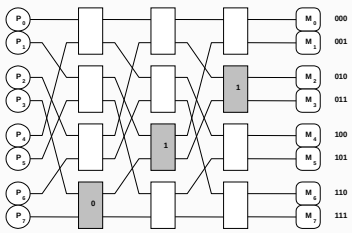
- Coherence misses
  - True sharing misses
    - Write to shared block (transmission of invalidation)
    - Read an invalidated block
  - False sharing misses
    - Read an unmodified word in an invalidated block

Revisit: Coherency Solutions

- Snooping Solution (Snoopy Bus):
  - Send all requests for data to all caches
  - Requires broadcast, works well with bus (natural broadcast medium)
  - Dominates for small scale machines (most of the market)
- Directory-Based Schemes
  - Keep track of what is being shared in 1 centralized place (logically)
  - Send point-to-point requests to processors via network
  - Scales better than Snooping

Scalable Shared Memory Architectures

Used in IBM SP Multiprocessor



220