

COMPUTER ORGANIZATION AND DESIGN

The Hardware/Software Interface



Chapter 5

Large and Fast: Exploiting Memory Hierarchy

Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
 - Items accessed recently are likely to be accessed again soon
 - e.g., instructions in a loop, induction variables
- Spatial locality
 - Items near those accessed recently are likely to be accessed soon
 - E.g., sequential instruction access, array data

Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
 - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
 - Cache memory attached to CPU



bj8sgs2poom1 bj8sgs2poom1 bj8sgs2poom1 bj8sgs2poom1 emory Hierarchy Levels



- Block (aka line): unit of copying
 - May be multiple words
- If accessed data is present in upper level
 - Hit: access satisfied by upper level
 - Hit ratio: hits/accesses
- If accessed data is absent
 - Miss: block copied from lower level
 - Time taken: miss penalty
 - Miss ratio: misses/accesses
 = 1 hit ratio
 - Then accessed data supplied from upper level



A Typical Memory Hierarchy

Take advantage of the principle of locality to present the user with as much memory as is available in the *cheapest* technology at the speed offered by the *fastest* technology



Memory Technology

- Static RAM (SRAM)
 - 0.5ns 2.5ns, \$2000 \$5000 per GB
- Dynamic RAM (DRAM)
 - 50ns 70ns, \$20 \$75 per GB
- Magnetic disk
 - 5ms 20ms, \$0.20 \$2 per GB
- Ideal memory
 - Access time of SRAM
 - Capacity and cost/GB of disk



DRAM Technology

Data stored as a charge in a capacitor

- Single transistor used to access the charge
- Must periodically be refreshed
 - Read contents and write back
 - Performed on a DRAM "row"





Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
 - DRAM accesses an entire row
 - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM
 - Transfer on rising and falling clock edges
- Quad data rate (QDR) DRAM
 - Separate DDR inputs and outputs



DRAM Generations

Year	Capacity	\$/GB
1980	64Kbit	\$1500000
1983	256Kbit	\$500000
1985	1Mbit	\$200000
1989	4Mbit	\$50000
1992	16Mbit	\$15000
1996	64Mbit	\$10000
1998	128Mbit	\$4000
2000	256Mbit	\$1000
2004	512Mbit	\$250
2007	1Gbit	\$50





DRAM Performance Factors

Row buffer

- Allows several words to be read and refreshed in parallel
- Synchronous DRAM
 - Allows for consecutive accesses in bursts without needing to send each address
 - Improves bandwidth
- DRAM banking
 - Allows simultaneous access to multiple DRAMs
 - Improves bandwidth



Increasing Memory Bandwidth



Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

Flash Storage

Nonvolatile semiconductor storage

- 100× 1000× faster than disk
- Smaller, lower power, more robust
- But more \$/GB (between disk and DRAM)







Flash Types

- NOR flash: bit cell like a NOR gate
 - Random read/write access
 - Used for instruction memory in embedded systems
- NAND flash: bit cell like a NAND gate
 - Denser (bits/area), but block-at-a-time access
 - Cheaper per GB
 - Used for USB keys, media storage, …
- Flash bits wears out after 1000's of accesses
 - Not suitable for direct RAM or disk replacement
 - Wear leveling: remap data to less used blocks





Nonvolatile, rotating magnetic storage







Chapter 6 — Storage and Other I/O Topics — 14

Disk Sectors and Access

- Each sector records
 - Sector ID
 - Data (512 bytes, 4096 bytes proposed)
 - Error correcting code (ECC)
 - Used to hide defects and recording errors
 - Synchronization fields and gaps
- Access to a sector involves
 - Queuing delay if other accesses are pending
 - Seek: move the heads
 - Rotational latency
 - Data transfer
 - Controller overhead

Disk Access Example

- Given
 - 512B sector, 15,000rpm, 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk
- Average read time
 - 4ms seek time + ½ / (15,000/60) = 2ms rotational latency + 512 / 100MB/s = 0.005ms transfer time + 0.2ms controller delay = 6.2ms
- If actual average seek time is 1ms
 - Average read time = 3.2ms



Disk Performance Issues

- Manufacturers quote average seek time
 - Based on all possible seeks
 - Locality and OS scheduling lead to smaller actual average seek times
- Smart disk controller allocate physical sectors on disk
 - Present logical sector interface to host
 - SCSI, ATA, SATA
- Disk drives include caches
 - Prefetch sectors in anticipation of access
 - Avoid seek and rotational delay



Cache Memory

- Cache memory
 - The level of the memory hierarchy closest to the CPU
 - Given accesses $X_1, \ldots, X_{n-1}, X_n$





a. Before the reference to X_n

b. After the reference to X_n

- How do we know if the data is present?
- Where do we look?



Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
 - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits



Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
 - Store block address as well as the data
 - Actually, only need the high-order bits
 - Called the tag
- What if there is no data in a location?
 - Valid bit: 1 = present, 0 = not present
 - Initially 0



8-blocks, 1 word/block, direct mappedInitial state

Index	V	Тад	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		



Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Υ	10	Mem[10110]
111	N		



Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	Υ	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		



Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		



Word a	ddr	Binary ad	dr	Hit/miss	Cache block
16		10 000		Miss	000
3		00 011		Miss	011
16		10 000		Hit	000
Index	V	Tag	Dat	а	
000	Υ	10	Mem[10000]		
001	N				
010	Y	11	Mei	m[11010]	
011	Υ	00	Me	m [00011]	
100	N				
101	N				
110	Υ	10	Mei	m[10110]	
111	N				



Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Υ	10	Mem[10010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		



Address Subdivision





The Memory Hierarchy: Terminology

- Block (or line): the minimum unit of information that is present (or not) in a cache
- Hit Rate: the fraction of memory accesses found in a level of the memory hierarchy
 - Hit Time: Time to access that level which consists of

Time to access the block + Time to determine hit/miss

- □ Miss Rate: the fraction of memory accesses *not* found in a level of the memory hierarchy \Rightarrow 1 (Hit Rate)
 - Miss Penalty: Time to replace a block in that level with the corresponding block from a lower level which consists of

Time to access the block in the lower level + Time to transmit that block to the level that experienced the miss + Time to insert the block in that level + Time to pass the block to the requestor

Hit Time << Miss Penalty

Characteristics of the Memory Hierarchy



(Relative) size of the memory at each level

How is the Hierarchy Managed?

 \Box registers \leftrightarrow memory

by compiler (programmer?)

 \Box cache \leftrightarrow main memory

- by the cache controller hardware
- \Box main memory \leftrightarrow disks
 - by the operating system (virtual memory)
 - virtual to physical address mapping assisted by the hardware (TLB)
 - by the programmer (files)

Cache Basics

- □ Two questions to answer (in hardware):
 - C Q1: How do we know if a data item is in the cache?
 - C Q2: If it is, how do we find it?
- Direct mapped
 - E Each memory block is mapped to exactly one block in the cache
 - lots of lower level blocks must share blocks in the cache
 - Address mapping (to answer Q2):
 (block address) modulo (# of blocks in the cache)
 - Have a tag associated with each cache block that contains the address information (the upper portion of the address) required to identify the block (to answer Q1)

Caching: A Simple First Example



One word blocks Two low order bits define the byte in the word (32b words)

Q2: How do we find it?

Use next 2 low order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

(block address) modulo (# of blocks in the cache)

is in the cache

Direct Mapped Cache

Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 miss 1 miss 2 miss 3 miss Mem(0)00 Mem(0)00 00 Mem(0)00 Mem(0)00 Mem(1)Mem(1)00 00 <u>Mem(1)</u> 00 Mem(2)Mem(2) 00 00 Mem(3)hit **15** miss 4 miss 3 hit 4 Mem(0) Mem(4)00 01 Mem(4)01 Mem(4)01 Mem(1)Mem(1)Mem(1)Mem(1)00 00 00 00 Mem(2) Mem(2)Mem(2) Mem(2) 00 00 00 00

00

Mem(3)

0 1 2 3 4 3 4

15

8 requests, 6 misses

00

Mem(3)

Mem(3)

00

()

15

Mem(3)

00

11

MIPS Direct Mapped Cache Example

One word blocks, cache size = 1K words (or 4KB)



What kind of locality are we taking advantage of?

Multiword Block Direct Mapped Cache

□ Four words/block, cache size = 1K words



What kind of locality are we taking advantage of?

CSE431 Chapter 5A.37

Taking Advantage of Spatial Locality

Let cache block hold more than one word

Start with an empty cache - all blocks initially marked as not valid



 $\mathbf{0}$

1 2 3 4 3

4

15

4 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)



8 requests, 4 misses
Miss Rate vs Block Size vs Cache Size



Block size (bytes)

Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing capacity misses)

Cache Field Sizes

- The number of bits in a cache includes both the storage for data and for the tags
 - □ 32-bit byte address
 - For a direct mapped cache with 2ⁿ blocks, n bits are used for the index
 - For a block size of 2^m words (2^{m+2} bytes), *m* bits are used to address the word within the block and 2 bits are used to address the byte within the word
- □ What is the size of the tag field?
- The total number of bits in a direct-mapped cache is then 2ⁿ x (block size + tag field size + valid field size)
- How many total bits are required for a direct mapped cache with 16KB of data and 4-word blocks assuming a 32-bit address?

Handling Cache Hits

- Read hits (I\$ and D\$)
 - this is what we want!
- Write hits (D\$ only)
 - require the cache and memory to be consistent
 - always write the data into both the cache block and the next level in the memory hierarchy (write-through)
 - writes run at the speed of the next level in the memory hierarchy so slow! – or can use a write buffer and stall only if the write buffer is full
 - allow cache and memory to be inconsistent
 - write the data only into the cache block (write-back the cache block to the next level in the memory hierarchy when that cache block is "evicted")
 - need a dirty bit for each data cache block to tell if it needs to be written back to memory when it is evicted – can use a write buffer to help "buffer" write-backs of dirty blocks

Sources of Cache Misses

- Compulsory (cold start or process migration, first reference):
 - First access to a block, "cold" fact of life, not a whole lot you can do about it. If you are going to run "millions" of instruction, compulsory misses are insignificant
 - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)

□ Capacity:

- Cache cannot contain all blocks accessed by the program
- Solution: increase cache size (may increase access time)

Conflict (collision):

- Multiple memory locations mapped to the same cache location
- Solution 1: increase cache size
- Solution 2: increase associativity (stay tuned) (may increase access time)

Handling Cache Misses (Single Word Blocks)

Read misses (I\$ and D\$)

stall the pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send the requested word to the processor, then let the pipeline resume

Write misses (D\$ only)

1. stall the pipeline, fetch the block from next level in the memory hierarchy, install it in the cache (which may involve having to evict a dirty block if using a write-back cache), write the word from the processor to the cache, then let the pipeline resume

or

2. Write allocate – just write the word into the cache updating both the tag and data, no need to check for cache hit, no need to stall

or

3. No-write allocate – skip the cache write (but must invalidate that cache block since it will now hold stale data) and just write the word to the write buffer (and eventually to the next memory level), no need to stall if the write buffer isn't full

CSE431 Chapter 5A.44

Irwin, PSU, 2008

Multiword Block Considerations

Read misses (I\$ and D\$)

- Processed the same as for single word blocks a miss returns the entire block from memory
- Miss penalty grows as block size grows
 - Early restart processor resumes execution as soon as the requested word of the block is returned
 - Requested word first requested word is transferred from the memory to the cache (and processor) first
- Nonblocking cache allows the processor to continue to access the cache while the cache is handling an earlier miss

Write misses (D\$)

If using write allocate must *first* fetch the block from memory and then write the word to the block (or could end up with a "garbled" block in the cache (e.g., for 4 word blocks, a new tag, one word of data from the new block, and three words of data from the old block)

Memory Systems that Support Caches

The off-chip interconnect and memory architecture can affect overall system performance in dramatic ways



One word wide organization (one word wide bus and one word wide memory)

- Assume
 - 1. 1 memory bus clock cycle to send the addr
 - 15 memory bus clock cycles to get the 1st word in the block from DRAM (row cycle time), 5 memory bus clock cycles for 2nd, 3rd, 4th words (column access time)
 - 3. 1 memory bus clock cycle to return a word of data
 - Memory-Bus to Cache bandwidth
 - number of bytes accessed from memory and transferred to cache/CPU per memory bus clock cycle

Example: Larger Block Size

- 64 blocks, 16 bytes/block
 - To what block number does address 1200 map?
- Block address = $\lfloor 1200/16 \rfloor = 75$
- Block number = 75 modulo 64 = 11





Block Size Considerations

- Larger blocks should reduce miss rate
 - Due to spatial locality
- But in a fixed-sized cache
 - Larger blocks \Rightarrow fewer of them
 - More competition \Rightarrow increased miss rate
 - Larger blocks \Rightarrow pollution
- Larger miss penalty
 - Can override benefit of reduced miss rate
 - Early restart and critical-word-first can help



Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
 - Stall the CPU pipeline
 - Fetch block from next level of hierarchy
 - Instruction cache miss
 - Restart instruction fetch
 - Data cache miss
 - Complete data access



Write-Through

- On data-write hit, could just update the block in cache
 - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
 - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
 - Effective CPI = 1 + 0.1×100 = 11
- Solution: write buffer
 - Holds data waiting to be written to memory
 - CPU continues immediately
 - Only stalls on write if write buffer is already full



Write-Back

- Alternative: On data-write hit, just update the block in cache
 - Keep track of whether each block is dirty
- When a dirty block is replaced
 - Write it back to memory
 - Can use a write buffer to allow replacing block to be read first



Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
 - Allocate on miss: fetch the block
 - Write around: don't fetch the block
 - Since programs often write a whole block before reading it (e.g., initialization)
 - For write-back
 - Usually fetch the block



Example: Intrinsity FastMATH

- Embedded MIPS processor
 - 12-stage pipeline
 - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
 - Each 16KB: 256 blocks × 16 words/block
 - D-cache: write-through or write-back
 - SPEC2000 miss rates
 - I-cache: 0.4%
 - D-cache: 11.4%
 - Weighted average: 3.2%



Example: Intrinsity FastMATH



Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 54

 \leq

Main Memory Supporting Caches

- Use DRAMs for main memory
 - Fixed width (e.g., 1 word)
 - Connected by fixed-width clocked bus
 - Bus clock is typically slower than CPU clock
- Example cache block read
 - 1 bus cycle for address transfer
 - 15 bus cycles per DRAM access
 - 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
 - Miss penalty = $1 + 4 \times 15 + 4 \times 1 = 65$ bus cycles
 - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle



Measuring Cache Performance

Components of CPU time
Program execution cycles
Includes cache hit time
Memory stall cycles
Mainly from cache misses
With simplifying assumptions:

Memory stall cycles

= Memory accesses Program × Miss rate × Miss penalty

 $= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$



Cache Performance Example

- Given
 - I-cache miss rate = 2%
 - D-cache miss rate = 4%
 - Miss penalty = 100 cycles
 - Base CPI (ideal cache) = 2
 - Load & stores are 36% of instructions
- Miss cycles per instruction
 - I-cache: 0.02 × 100 = 2
 - D-cache: 0.36 × 0.04 × 100 = 1.44
- Actual CPI = 2 + 2 + 1.44 = 5.44

Ideal CPU is 5.44/2 =2.72 times faster



Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
 - AMAT = Hit time + Miss rate × Miss penalty
- Example
 - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
 - AMAT = 1 + 0.05 × 20 = 2ns
 - 2 cycles per instruction



Performance Summary

- When CPU performance increased
 - Miss penalty becomes more significant
- Decreasing base CPI
 - Greater proportion of time spent on memory stalls
- Increasing clock rate
 - Memory stalls account for more CPU cycles
- Can't neglect cache behavior when evaluating system performance



Associative Caches

Fully associative

- Allow a given block to go in any cache entry
- Requires all entries to be searched at once
- Comparator per entry (expensive)
- n-way set associative
 - Each set contains n entries
 - Block number determines which set
 - Block number) modulo (#Sets in cache)
 - Search all entries in a given set at once
 - n comparators (less expensive)



Associative Cache Example





Spectrum of Associativity

For a cache with 8 entries

One-way set associative



Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

 Tag
 Data
 Data
 Tag
 Data
 Tag
 Data
 Tag
 Data
 Tag
 <thData</th>
 <thData</th>
 <thData</th>
 <thData</th>



Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 62

Associativity Example

- Compare 4-block caches
 - Direct mapped, 2-way set associative, fully associative
 - Block access sequence: 0, 8, 0, 6, 8

Direct mapped

Block	Cache	Hit/miss	Cache content after access				
address	index		0	1	2	3	
0	0	miss	Mem[0]				
8	0	miss	Mem[8]				
0	0	miss	Mem[0]				
6	2	miss	Mem[0]		Mem[6]		
8	0	miss	Mem[8]		Mem[6]		



Associativity Example

2-way set associative

Block	Cache	Hit/miss	Cache content after access				
address	index		Set 0		Se	t 1	
0	0	miss	Mem[0]				
8	0	miss	Mem[0]	Mem[8]			
0	0	hit	Mem[0]	Mem[8]			
6	0	miss	Mem[0]	Mem[6]			
8	0	miss	Mem[8]	Mem[6]			

Fully associative

Block address	Hit/miss	Cache content after access				
0	miss	Mem[0]				
8	miss	Mem[0]	Mem[8]			
0	hit	Mem[0]	Mem[8]			
6	miss	Mem[0]	Mem[8]	Mem[6]		
8	hit	Mem[0]	Mem[8]	Mem[6]		



How Much Associativity

- Increased associativity decreases miss rate
 - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
 - 1-way: 10.3%
 - 2-way: 8.6%
 - 4-way: 8.3%
 - 8-way: 8.1%



Set Associative Cache Organization





Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 66

Average Memory Access Time (AMAT)

- A larger cache will have a longer access time. An increase in hit time will likely add another stage to the pipeline. At some point the increase in hit time for a larger cache will overcome the improvement in hit rate leading to a decrease in performance.
- Average Memory Access Time (AMAT) is the average to access memory considering both hits and misses

AMAT = Time for a hit + Miss rate x Miss penalty

What is the AMAT for a processor with a 20 psec clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache access time of 1 clock cycle?

Reducing Cache Miss Rates #1

1. Allow more flexible block placement

- In a direct mapped cache a memory block maps to exactly one cache block
- At the other extreme, could allow a memory block to be mapped to any cache block – fully associative cache

A compromise is to divide the cache into sets each of which consists of n "ways" (n-way set associative). A memory block maps to a unique set (specified by the index field) and can be placed in any way of that set (so there are n choices)

(block address) modulo (# sets in the cache)

Another Reference String Mapping

Consider the main memory word reference string

 $\mathbf{0}$

4

0

Start with an empty cache - all blocks initially marked as not valid



4

 $\mathbf{0}$

4

 $\mathbf{0}$

4

8 requests, 8 misses

Ping pong effect due to conflict misses - two memory locations that map into the same cache block

CSE431 Chapter 5A.70

Irwin, PSU, 2008

Set Associative Cache Example



Main Memory

One word blocks Two low order bits define the byte in the word (32b words)

Q2: How do we find it?

Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache)

Another Reference String Mapping

Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid



0 4 0 4 0 4 0 4

- 1 8 requests, 2 misses
- Solves the ping pong effect in a direct mapped cache due to conflict misses since now two memory locations that map into the same cache set can co-exist!

Four-Way Set Associative Cache

 \square 2⁸ = 256 sets each with four ways (each with one block)



Range of Set Associative Caches

For a fixed size cache, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number or ways) and halves the number of sets – decreases the size of the index by 1 bit and increases the size of the tag by 1 bit



Costs of Set Associative Caches

- When a miss occurs, which way's block do we pick for replacement?
 - Least Recently Used (LRU): the block replaced is the one that has been unused for the longest time
 - Must have hardware to keep track of when each way's block was used relative to the other blocks in the set
 - For 2-way set associative, takes one bit per set → set the bit when a block is referenced (and reset the other way's bit)
- N-way set associative cache costs
 - N comparators (delay and area)
 - MUX delay (set selection) before data is available
 - Data available after set selection (and Hit/Miss decision). In a direct mapped cache, the cache block is available before the Hit/Miss decision
 - So its not possible to just assume a hit and continue and recover later if it was a miss

Benefits of Set Associative Caches

The choice of direct mapped or set associative depends on the cost of a miss versus the cost of implementation



Data from Hennessy & Patterson, *Computer Architecture*, 2003

Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)
Reducing Cache Miss Rates #2

- 2. Use multiple levels of caches
- With advancing technology have more than enough room on the die for bigger L1 caches or for a second level of caches – normally a unified L2 cache (i.e., it holds both instructions and data) and in some cases even a unified L3 cache
- For our example, CPI_{ideal} of 2, 100 cycle miss penalty (to main memory) and a 25 cycle miss penalty (to UL2\$), 36% load/stores, a 2% (4%) L1 I\$ (D\$) miss rate, add a 0.5% UL2\$ miss rate

$$CPI_{stalls} = 2 + .02 \times 25 + .36 \times .04 \times 25 + .005 \times 100 + .36 \times .005 \times 100 = 3.54$$

(as compared to 5.44 with no L2\$)

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 with smaller block sizes
 - Secondary cache(s) should focus on reducing miss rate to reduce the penalty of long main memory access times
 - Larger with larger block sizes
 - Higher levels of associativity
- The miss penalty of the L1 cache is significantly reduced by the presence of an L2 cache – so it can be smaller (i.e., faster) but have a higher miss rate
- □ For the L2 cache, hit time is less important than miss rate
 - The L2\$ hit time determines L1\$'s miss penalty
 - L2\$ local miss rate >> than the global miss rate

Two Machines' Cache Parameters

	Intel Nehalem	AMD Barcelona				
L1 cache organization & size	Split I\$ and D\$; 32KB for each per core; 64B blocks	Split I\$ and D\$; 64KB for each per core; 64B blocks				
L1 associativity	4-way (I), 8-way (D) set assoc.; ~LRU replacement	2-way set assoc.; LRU replacement				
L1 write policy	write-back, write-allocate	write-back, write-allocate				
L2 cache organization & size	Unified; <mark>256KB</mark> (0.25MB) per core; 64B blocks	Unified; 512KB (0.5MB) per core; 64B blocks				
L2 associativity	8-way set assoc.; ~LRU	16-way set assoc.; ~LRU				
L2 write policy	write-back	write-back				
L2 write policy	write-back, write-allocate	write-back, write-allocate				
L3 cache organization & size	Unified; 8192KB (8MB) shared by cores; 64B blocks	Unified; 2048KB (2MB) shared by cores; 64B blocks				
L3 associativity	16-way set assoc.	32-way set assoc.; evict block shared by fewest cores				
L3 write policy	write-back, write-allocate	write-back; write-allocate				

FSM Cache Controller

Key characteristics for a simple L1 cache

- I Direct mapped
- Write-back using write-allocate
- Block size of 4 32-bit words (so 16B); Cache size of 16KB (so 1024 blocks)
- 18-bit tags, 10-bit index, 2-bit block offset, 2-bit byte offset, dirty bit, valid bit, LRU bits (if set associative)

Processor



Four State Cache Controller



Replacement Policy

- Direct mapped: no choice
- Set associative
 - Prefer non-valid entry, if there is one
 - Otherwise, choose among entries in the set
- Least-recently used (LRU)
 - Choose the one unused for the longest time
 - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
 - Gives approximately the same performance as LRU for high associativity



Multilevel Caches

- Primary cache attached to CPU
 - Small, but fast
- Level-2 cache services misses from primary cache
 - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache



Multilevel Cache Example

- Given
 - CPU base CPI = 1, clock rate = 4GHz
 - Miss rate/instruction = 2%
 - Main memory access time = 100ns
- With just primary cache
 - Miss penalty = 100ns/0.25ns = 400 cycles
 - Effective CPI = 1 + 0.02 × 400 = 9



Example (cont.)

- Now add L-2 cache
 - Access time = 5ns
 - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
 - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
 - Extra penalty = 500 cycles
- $CPI = 1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$
- Performance ratio = 9/3.4 = 2.6



Multilevel Cache Considerations

- Primary cache
 - Focus on minimal hit time
- L-2 cache
 - Focus on low miss rate to avoid main memory access
 - Hit time has less overall impact
- Results
 - L-1 cache usually smaller than a single cache
 - L-1 block size smaller than L-2 block size



Interactions with Advanced CPUs

- Out-of-order CPUs can execute instructions during cache miss
 - Pending store stays in load/store unit
 - Dependent instructions wait in reservation stations
 - Independent instructions continue
- Effect of miss depends on program data flow
 - Much harder to analyse
 - Use system simulation



Interactions with Software

Misses depend on memory access patterns

Algorithm behaviorCompiler

optimization for memory access





Software Optimization via Blocking

- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM:

```
for (int j = 0; j < n; ++j)
{
    double cij = C[i+j*n];
    for( int k = 0; k < n; k++ )
        cij += A[i+k*n] * B[k+j*n];
    C[i+j*n] = cij;
}</pre>
```







Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 94

Cache Blocked DGEMM

```
1 #define BLOCKSIZE 32
2 void do block (int n, int si, int sj, int sk, double *A, double
3 *B, double *C)
4 {
5
  for (int i = si; i < si+BLOCKSIZE; ++i)</pre>
6
    for (int j = sj; j < sj+BLOCKSIZE; ++j)
7
  {
8
     double cij = C[i+j*n];/* cij = C[i][j] */
9
     for ( int k = sk; k < sk+BLOCKSIZE; k++ )
10
    cij += A[i+k*n] * B[k+j*n];/* cij+=A[i][k]*B[k][j] */
11
    C[i+j*n] = cij; /* C[i][j] = cij */
12 }
13 }
14 void dgemm (int n, double* A, double* B, double* C)
15 {
16
   for (int sj = 0; sj < n; sj += BLOCKSIZE )
     for ( int si = 0; si < n; si += BLOCKSIZE )</pre>
17
18
      for ( int sk = 0; sk < n; sk += BLOCKSIZE )</pre>
       do block(n, si, sj, sk, A, B, C);
19
20 }
```

Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 95

Blocked DGEMM Access Pattern





Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 96

Dependability



Fault: failure of a component

 May or may not lead to system failure



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



The Hamming SEC Code

- Hamming distance
 - Number of bits that are different between two bit patterns
- Minimum distance = 2 provides single bit error detection
 - E.g. parity code
- Minimum distance = 3 provides single error correction, 2 bit error detection



Encoding SEC

- To calculate Hamming code:
 - Number bits from 1 on the left
 - All bit positions that are a power 2 are parity bits
 - Each parity bit checks certain data bits:

Bit positi	on	1	2	3	4	5	6	7	8	9	10	11	12
Encoded date	bits	p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
Parity bit coverate	p1	Х		Х		Х		Х		Х		Х	
	p2		Х	Х			Х	Х			Х	Х	
	p4				Х	Х	Х	Х					Х
	p8								Х	Х	Х	Х	Х



Decoding SEC

- Value of parity bits indicates which bits are in error
 - Use numbering from encoding procedure
 - E.g.
 - Parity bits = 0000 indicates no error
 - Parity bits = 1010 indicates bit 10 was flipped



SEC/DEC Code

- Add an additional parity bit for the whole word (p_n)
- Make Hamming distance = 4

Decoding:

- Let H = SEC parity bits
 - H even, p_n even, no error
 - H odd, p_n odd, correctable single bit error
 - H even, p_n odd, error in p_n bit
 - H odd, p_n even, double error occurred
- Note: ECC DRAM uses SEC/DEC with 8 bits protecting each 64 bits



Virtual Machines

- Host computer emulates guest operating system and machine resources
 - Improved isolation of multiple guests
 - Avoids security and reliability problems
 - Aids sharing of resources
- Virtualization has some performance impact
 - Feasible with modern high-performance comptuers
 - Examples
 - IBM VM/370 (1970s technology!)
 - VMWare
 - Microsoft Virtual PC



Virtual Machine Monitor

- Maps virtual resources to physical resources
 - Memory, I/O devices, CPUs
- Guest code runs on native machine in user mode
 - Traps to VMM on privileged instructions and access to protected resources
- Guest OS may be different from host OS
- VMM handles real I/O devices
 - Emulates generic virtual I/O devices for guest



Example: Timer Virtualization

- In native machine, on timer interrupt
 - OS suspends current process, handles interrupt, selects and resumes next process
- With Virtual Machine Monitor
 - VMM suspends current VM, handles interrupt, selects and resumes next VM
- If a VM requires timer interrupts
 - VMM emulates a virtual timer
 - Emulates interrupt for VM when physical timer interrupt occurs



Instruction Set Support

- User and System modes
- Privileged instructions only available in system mode
 - Trap to system if executed in user mode
- All physical resources only accessible using privileged instructions
 - Including page tables, interrupt controls, I/O registers
- Renaissance of virtualization support
 - Current ISAs (e.g., x86) adapting



Virtual Memory

- Use main memory as a "cache" for secondary (disk) storage
 - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
 - Each gets a private virtual address space holding its frequently used code and data
 - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
 - VM "block" is called a page
 - VM translation "miss" is called a page fault



Address Translation

Fixed-size pages (e.g., 4K)



Physical address

Virtual address



Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 108

Page Fault Penalty

- On page fault, the page must be fetched from disk
 - Takes millions of clock cycles
 - Handled by OS code
- Try to minimize page fault rate
 - Fully associative placement
 - Smart replacement algorithms



Page Tables

- Stores placement information
 - Array of page table entries, indexed by virtual page number
 - Page table register in CPU points to page table in physical memory
- If page is present in memory
 - PTE stores the physical page number
 - Plus other status bits (referenced, dirty, ...)
- If page is not present
 - PTE can refer to location in swap space on disk



Translation Using a Page Table



Physical address



Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 111

Mapping Pages to Storage





Replacement and Writes

- To reduce page fault rate, prefer leastrecently used (LRU) replacement
 - Reference bit (aka use bit) in PTE set to 1 on access to page
 - Periodically cleared to 0 by OS
 - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
 - Block at once, not individual locations
 - Write through is impractical
 - Use write-back
 - Dirty bit in PTE set when page is written



Fast Translation Using a TLB

- Address translation would appear to require extra memory references
 - One to access the PTE
 - Then the actual memory access
- But access to page tables has good locality
 - So use a fast cache of PTEs within the CPU
 - Called a Translation Look-aside Buffer (TLB)
 - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
 - Misses could be handled by hardware or software



Fast Translation Using a TLB





TLB Misses

- If page is in memory
 - Load the PTE from memory and retry
 - Could be handled in hardware
 - Can get complex for more complicated page table structures
 - Or in software
 - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
 - OS handles fetching the page and updating the page table
 - Then restart the faulting instruction


TLB Miss Handler

- TLB miss indicates
 - Page present, but PTE not in TLB
 - Page not preset
- Must recognize TLB miss before destination register overwritten
 - Raise exception
- Handler copies PTE from memory to TLB
 - Then restarts instruction
 - If page not present, page fault will occur



Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
 - If dirty, write to disk first
 - Read page into memory and update page table
- Make process runnable again
 - Restart from faulting instruction



TLB and Cache Interaction



- If cache tag uses physical address
 - Need to translate before cache lookup
- Alternative: use virtual address tag
 - Complications due to aliasing
 - Different virtual addresses for shared physical address

Memory Protection

- Different tasks can share parts of their virtual address spaces
 - But need to protect against errant access
 - Requires OS assistance
- Hardware support for OS protection
 - Privileged supervisor mode (aka kernel mode)
 - Privileged instructions
 - Page tables and other state information only accessible in supervisor mode
 - System call exception (e.g., syscall in MIPS)



The Memory Hierarchy

The BIG Picture

- Common principles apply at all levels of the memory hierarchy
 - Based on notions of caching
- At each level in the hierarchy
 - Block placement
 - Finding a block
 - Replacement on a miss
 - Write policy



Block Placement

Determined by associativity

- Direct mapped (1-way associative)
 - One choice for placement
- n-way set associative
 - n choices within a set
- Fully associative
 - Any location
- Higher associativity reduces miss rate
 - Increases complexity, cost, and access time



Finding a Block

Associativity	Location method	Tag comparisons
Direct mapped	Index	1
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries	#entries
	Full lookup table	0

Hardware caches

Reduce comparisons to reduce cost

Virtual memory

- Full table lookup makes full associativity feasible
- Benefit in reduced miss rate



Replacement

Choice of entry to replace on a miss

- Least recently used (LRU)
 - Complex and costly hardware for high associativity
- Random
 - Close to LRU, easier to implement
- Virtual memory
 - LRU approximation with hardware support



Write Policy

- Write-through
 - Update both upper and lower levels
 - Simplifies replacement, but may require write buffer
- Write-back
 - Update upper level only
 - Update lower level when block is replaced
 - Need to keep more state
- Virtual memory
 - Only write-back is feasible, given disk write latency



Sources of Misses

- Compulsory misses (aka cold start misses)
 - First access to a block
- Capacity misses
 - Due to finite cache size
 - A replaced block is later accessed again
 - Conflict misses (aka collision misses)
 - In a non-fully associative cache
 - Due to competition for entries in a set
 - Would not occur in a fully associative cache of the same total size



Cache Design Trade-offs

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.



Cache Control

Example cache characteristics

- Direct-mapped, write-back, write allocate
- Block size: 4 words (16 bytes)
- Cache size: 16 KB (1024 blocks)
- 32-bit byte addresses
- Valid bit and dirty bit per block
- Blocking cache
 - CPU waits until access is complete





Interface Signals





Finite State Machines

- Use an FSM to sequence control steps
- Set of states, transition on each clock edge
 - State values are binary encoded
 - Current state stored in a register
 - Next state
 - = f_n (current state, current inputs)
- Control output signals = f_o (current state)



Cache Controller FSM





Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 131

Cache Coherence Problem

- Suppose two CPU cores share a physical address space
 - Write-through caches

Time step	Event	CPU A's cache	CPU B's cache	Memory
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A writes 1 to X	1	0	1



Coherence Defined

- Informally: Reads return most recently written value
- Formally:
 - P writes X; P reads X (no intervening writes)
 ⇒ read returns written value
 - P₁ writes X; P₂ reads X (sufficiently later)
 - \Rightarrow read returns written value
 - c.f. CPU B reading X after step 3 in example
 - P₁ writes X, P₂ writes X
 - \Rightarrow all processors see writes in the same order
 - End up with the same final value for X



Cache Coherence Protocols

- Operations performed by caches in multiprocessors to ensure coherence
 - Migration of data to local caches
 - Reduces bandwidth for shared memory
 - Replication of read-shared data
 - Reduces contention for access
- Snooping protocols
 - Each cache monitors bus reads/writes
- Directory-based protocols
 - Caches and memory record sharing status of blocks in a directory



Invalidating Snooping Protocols

- Cache gets exclusive access to a block when it is to be written
 - Broadcasts an invalidate message on the bus
 - Subsequent read in another cache misses

Owning cache supplies updated value

CPU activity	Bus activity	CPU A's cache	CPU B's cache	Memory
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes 1 to X	Invalidate for X	1		0
CPU B read X	Cache miss for X	1	1	1



Memory Consistency

- When are writes seen by other processors
 - "Seen" means a read returns the written value
 - Can't be instantaneously
- Assumptions
 - A write completes only when all processors have seen it
 - A processor does not reorder writes with other accesses
- Consequence
 - P writes X then writes Y
 - \Rightarrow all processors that see new Y also see new X
 - Processors can reorder reads, but not writes



Multilevel On-Chip Caches

Characteristic	ARM Cortex-A8	Intel Nehalem
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	4-way (I), 4-way (D) set associative	4-way (I), 8-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate(?)	Write-back, No-write-allocate
L1 hit time (load-use)	1 clock cycle	4 clock cycles, pipelined
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 1 MiB	256 KiB (0.25 MiB)
L2 cache associativity	8-way set associative	8-way set associative
L2 replacement	Random(?)	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate (?)	Write-back, Write-allocate
L2 hit time	11 clock cycles	10 clock cycles
L3 cache organization	-	Unified (instruction and data)
L3 cache size	-	8 MiB, shared
L3 cache associativity	-	16-way set associative
L3 replacement	-	Approximated LRU
L3 block size	-	64 bytes
L3 write policy	_	Write-back, Write-allocate
L3 hit time	_	35 clock cycles



2-Level TLB Organization

Characteristic	ARM Cortex-A8	Intel Core i7
Virtual address	32 bits	48 bits
Physical address	32 bits	44 bits
Page size	Variable: 4, 16, 64 KiB, 1, 16 MiB	Variable: 4 KiB, 2/4 MiB
TLB organization	1 TLB for instructions and 1 TLB for data	1 TLB for instructions and 1 TLB for data per core
	Both TLBs are fully associative,	Both L1 TLBs are four-way set
	with 32 entries, round robin replacement	associative, LRU replacement
	TLB misses handled in hardware	L1 I-TLB has 128 entries for small pages, 7 per thread for large pages
		L1 D-TLB has 64 entries for small pages, 32 for large pages
		The L2 TLB is four-way set associative, LRU replacement
		The L2 TLB has 512 entries
		TLB misses handled in hardware



Supporting Multiple Issue

- Both have multi-banked caches that allow multiple accesses per cycle assuming no bank conflicts
- Core i7 cache optimizations
 - Return requested word first
 - Non-blocking cache
 - Hit under miss
 - Miss under miss
 - Data prefetching



DGEMM

Combine cache blocking and subword parallelism





Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 140

Pitfalls

Byte vs. word addressing

- Example: 32-byte direct-mapped cache, 4-byte blocks
 - Byte 36 maps to block 1
 - Word 36 maps to block 4
- Ignoring memory system effects when writing or generating code
 - Example: iterating over rows vs. columns of arrays
 - Large strides result in poor locality



Pitfalls

- In multiprocessor with shared L2 or L3 cache
 - Less associativity than cores results in conflict misses
 - More cores \Rightarrow need to increase associativity
- Using AMAT to evaluate performance of out-of-order processors
 - Ignores effect of non-blocked accesses
 - Instead, evaluate performance by simulation



Pitfalls

- Extending address range using segments
 - E.g., Intel 80286
 - But a segment is not always big enough
 - Makes address arithmetic complicated
- Implementing a VMM on an ISA not designed for virtualization
 - E.g., non-privileged instructions accessing hardware resources
 - Either extend ISA, or require guest OS not to use problematic instructions



Concluding Remarks

- Fast memories are small, large memories are slow
 - We really want fast, large memories ⊗
 - Caching gives this illusion ③
- Principle of locality
 - Programs use a small part of their memory space frequently
- Memory hierarchy
 - L1 cache \leftrightarrow L2 cache $\leftrightarrow \dots \leftrightarrow$ DRAM memory \leftrightarrow disk
- Memory system design is critical for multiprocessors

