

Memory Hierarchy

Dr. Arjan Durrresi
Louisiana State University
Baton Rouge, LA 70810
Durrresi@Csc.LSU.Edu

These slides are available at:
http://www.csc.lsu.edu/~durrresi/CSC3501_07/



- Basics of Caches
- Measuring and Improving Cache Performance
- Framework for Memory Hierarchies

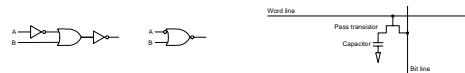
Memory

"You can never be too rich, too thin, or have too much memory"

- Create the illusion of unlimited fast memory
- Analogy with the use of library books
 - Get the books you need once
 - Or go to library each time you need new data
- 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
- Locality comes from natural program structure such as loops, arrays etc.
- Memory hierarchy: multiple levels of memory with different speeds and sizes

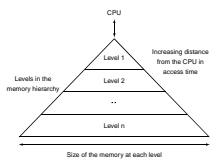
Memories: Review

- SRAM:
 - value is stored on a pair of inverting gates
 - very fast but takes up more space than DRAM (4 to 6 transistors)
- DRAM:
 - value is stored as a charge on capacitor (must be refreshed)
 - very small but slower than SRAM (factor of 5 to 10)



Exploiting Memory Hierarchy

- Users want large and fast memories!
- SRAM access times are 5 - 5ns at cost of \$4000 to \$10,000 per GB.
DRAM access times are 50-70ns at cost of \$100 to \$200 per GB - 2004.
Disk access times are 5 to 20 million ns at cost of \$.50 to \$2 per GB.
- Try and give it to them anyway
 - build a memory hierarchy



Basic Structure of Memory Hierarchy

Speed	CPU	Size	Cost (\$/bit)	Current Technology
Fastest	Memory	Smallest	Highest	SRAM
	Memory			DRAM
Slowest	Memory	Biggest	Lowest	Magnetic Disk

The user has the illusion of a memory as large as the largest level, But can be accessed as if it were all built from the fastest one

Locality

- A principle that makes having a memory hierarchy a good idea
- If an item is referenced,
 - temporal locality: it will tend to be referenced again soon
 - spatial locality: nearby items will tend to be referenced soon.

Why does code have locality?

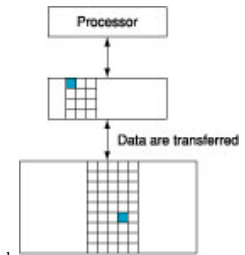
- Our initial focus: two levels (upper, lower)
 - block: minimum unit of data
 - hit: data requested is in the upper level
 - miss: data requested is not in the upper level

Memory Hierarchy

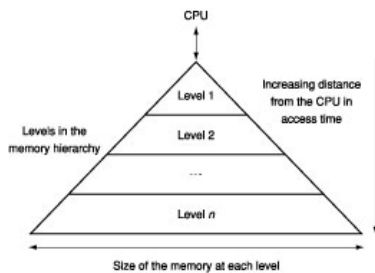
Miss rate: the fraction of memory Access not found in a level of the Memory hierarchy

Hit time: the time required to Access a level of the hierarchy, Including the time needed to determine The access is a hit or a miss

Miss penalty: the time required to fetch a block into a level of the memory Hierarchy from the lower level, including the time to access the block, transmit it, and insert in the needed level



Memory Hierarchy



Cache

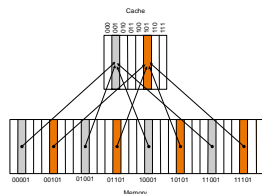
- Two issues:
 - How do we know if a data item is in the cache?
 - If it is, how do we find it?
- Our first example:
 - block size is one word of data
 - "direct mapped"

For each item of data at the lower level, there is exactly one location in the cache where it might be.

e.g., lots of items at the lower level share locations in the upper level

Direct Mapped Cache

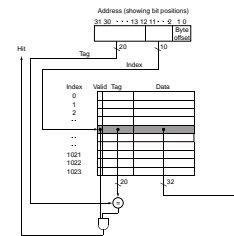
- Mapping: address is modulo the number of blocks in the cache



(Block Address) modulo (Number of cache blocks in the cache)

Direct Mapped Cache

- For MIPS:



Index - used to select the word

Tag - used to compare with the value of tag field of the cache

Valid bit - indicate whether an entry contains a valid address

Cache

- How many bits are required for a direct-mapped cache with 16 KB of data and 4-word blocks, assuming 32-bit address?
- 16 KB \rightarrow 4Kwords, 2^{12} words, with a block size of 4 words $\rightarrow 2^{10}$ blocks
- Each block has $4 \times 32 = 128$ bits data plus a tag
- If number of blocks = 2^n , word = 2^m , the tag field = $32 - n - m - 2$
 - Tag = $32 - 10 - 2 - 2 = 18$
- Total cache = $2^{10} \times (128 + (32 - 10 - 2 - 2) \times 1) = 2^{10} \times 147 = 147 \text{ Kbits} = 18.4 \text{ KB}$
- So 18.4 KB for a 16 KB cache, total number of bits is 1.15 times as needed just for the storage of data

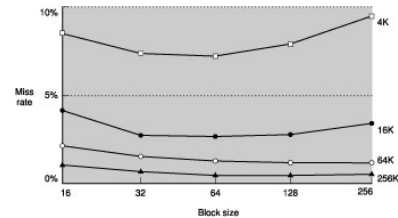
Mapping to Cache

- A cache with 64 blocks and a block size of 16 bytes.
- What block number does byte address 1200 map to?
- (Block Address) modulo (Number of cache blocks in the cache)
- The address of the block = $\frac{\text{Byte address}}{\text{Bytes per block}} = 75$
- That maps to cache block number (75 modulo 64) = 11
- This block maps all addresses between 1200 and 1215
- Larger blocks exploit spatial locality to lower miss rate
- But when block too big relatively to cache - more competition for those blocks - the block will be out of cache before many of its words are used

Block Size

- Increasing the block size - cost of a miss increases
- Miss penalty - time required to fetch the block from the next level of hierarchy and load it into the cache.
- Time to fetch
 - Latency to the first word
 - Transfer time for the rest of the block
- When the block size increases
 - The improvement in the miss rate starts to decrease
 - The increase in miss penalty > improvement in miss rate
 - The cache performance decreases

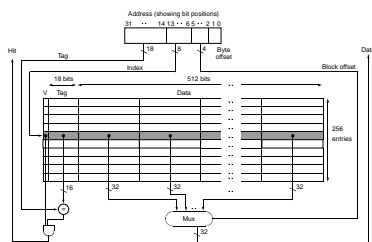
Miss Rate vs. Block Size



Each line represents a cache of different size
The miss rate goes up if the block size is too large relative to cache size.

Direct Mapped Cache

- Taking advantage of spatial locality:



For SPEC200 : Instruction miss rate = 0.4%, Data miss rate = 11.4%,
Effective combined miss rate = 3.2%

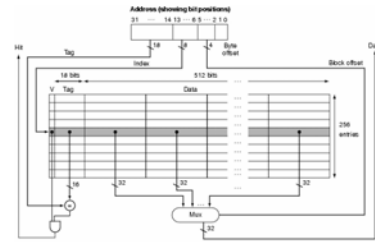
Hits vs. Misses

- Read hits
 - this is what we want!
- Read misses
 - stall the CPU, fetch block from memory, deliver to cache, restart
 - Similar to pipeline stalls. What is the difference among them?
- Steps for cache (instruction) miss :
 - Send the original PC value (current PC -4) to the memory
 - Instruct main memory to perform a read and wait for the memory to complete access.
 - Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address from ALU into the tag field, and turning the valid bit on.
 - Restart the instruction execution at the first step, which will refetch the instruction, this time in the cache.

Writes

- Write hits:
 - Avoid inconsistency between cache and memory
 - can replace data in cache and memory (write-through)
 - Writes very slow - 100 processor clock
 - SPEC2000 integer - 10% of instructions are stores
 - If CPI without writes is 1.0, spending 100 extra cycles on every writes: $1.0 + 100 \times 10\% = 11$
 - Use write in cache and in buffer - a write buffer stores the data while it is waiting to be written into memory.
 - If buffer is full - processor will stall
 - To avoid stalling - increase the depth of the buffer
 - write the data only into the cache (write-back the cache later). It written into memory when replaced. More complex mechanism.
- Write misses:
 - read the entire block into the cache, then write the word
 - Write the word into memory
 - Write-back
 - Uses a store buffer. Two cycles - one to write in the buffer, second to write from the buffer to the cache.
 - Write-through - can be done in one cycle

FastMATH Example



FastMATH - a fast embedded microprocessor - MIPS architecture
It has separate instruction and data cache. Each cache is 16KB, or 4K words, with 16-word blocks.

Read

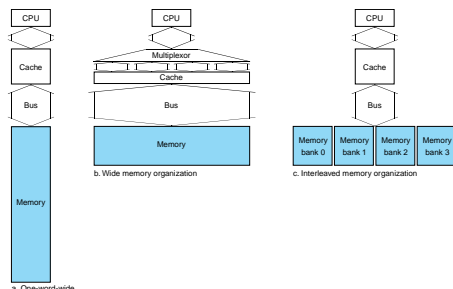
- Send address to the appropriate cache. The address comes from the PC (for an instruction) or from ALU (for data)
- If it is a hit, the requested word is available on the data lines. Since there are 16 words in the desired block, it is selected using a multiplexor.
- If it is a miss, the address is sent to the main memory. When the memory returns the data, it is written into the cache and then read.

Writes

- FastMATH offers both write-through and write-back
- Operating systems decide which strategy to use for a given application
- It has a one-entry write buffer
- Miss rate for SPEC2000 benchmarks:
 - Instruction miss rate = 0.4%
 - Data miss rate = 11.4%
 - Effective combined miss rate = 3.2% . Depends on the frequency of data and instructions rate.

Hardware Issues

- Make reading multiple words easier by using banks of memory

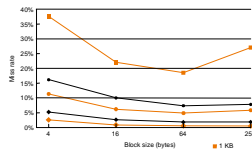


Access Time

- 1 memory bus clock cycle to send the address
- 15 memory bus clock for each DRAM access initiated
- 1 memory bus clock cycle to send a word data
- If we have a cache of four words:
 - One-word-wide memory - Miss penalty = $1 + 4 \times 15 + 4 \times 1 = 65$ memory bus clock cycles
 - Number of bytes transferred per bus clock for a single miss $4 \times 4 / 65 = .25$
 - With 2 word width (Fig b, slide 18) miss penalty = $1 + 2 \times 15 + 2 \times 1 = 33$
 - The bandwidth of a single miss is 0.48
 - With 4 word width (Fig b) miss penalty = $1 + 15 + 1 = 17$
 - The bandwidth of a single miss is 0.94
- Try to have access initiated in parallel, use banks of memories (Fig c, slide 19) Interleaving scheme:
 - Miss penalty = $1 + 1 \times 15 + 4 \times 1 = 20$ memory bus clock cycles
 - This is an effective bandwidth per miss of 0.80 bytes per clock, or about

Performance

- Increasing the block size tends to decrease miss rate:



Use split caches because there is more spatial locality in code:

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spicb	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

Performance

- Simplified model:

$$\text{execution time} = (\text{execution cycles} + \text{Memory-stall cycles}) \times \text{cycle time}$$

$$\text{Memory-stall cycles} = \# \text{ of instructions} \times \text{miss ratio} \times \text{miss penalty}$$

- Memory-stall = Read-stall cycles + Write-stall cycles
- Read-stall cycles = Reads/Program \times Read miss rate \times Read miss penalty
- Writes are more complex:
 - Write-through: write misses and buffer stalls (when buffer is full)
 - Write-stall cycles = Write/Program \times Write miss rate \times Write miss penalty + Write buffer stalls
 - With a deep buffer (e.g. four or more words) and a memory of accepting writes at a rate higher than the average program write frequency - write buffer stall can be ignored

Example of Cache Performance

- Assume an instruction miss rate for a program is 2% and a data cache miss rate is 4%
- If CPI is 2 without any memory stalls
- Miss penalty is 100 cycles for all misses
- How much faster it would run using SPECint2000 without miss?
- The number of memory miss cycles for instructions in terms of instructions I is :
 - Instruction miss cycle = $I \times 2\% \times 100 = 2.00 \times I$
- The frequency of stores in SPECint2000 is 36%:
 - Data miss cycles = $I \times 36\% \times 4\% \times 100 = 1.44 \times I$
- Total number of memory-stall cycles is $2I + 1.44I = 3.44I$
- CPI = $2 + 3.44 = 5.44$
- Performance improvement = $5.44/2 = 2.72$, amount of time in memory stalls = $3.44/5.44 = 63\%$
- If we speed up the processor CPI = 1, Performance improvement = 4.44, amount of time in memory stalls = $3.44/4.44 = 77\%$

Example (cont.)

- If we increase the processor clock rate by 2
- The miss penalty will be twice as many clocks - 200 clock cycles:
 - Total miss per instruction = $2\% \times 200 + 36\% \times 4\% \times 200 = 6.88$
- The faster compute will have CPI = $2 + 6.88 = 8.88$
- Let us compare the two computers:

$$\frac{\text{Performance fast clock}}{\text{Performance slow clock}} = \frac{\text{Execution time slow clock}}{\text{Execution time fast clock}}$$

$$= \frac{IC \times CPI_{\text{slow}} \times \text{Clock cycles}}{IC \times CPI_{\text{fast}} \times \frac{\text{Clock cycles}}{2}} = \frac{5.44}{8.88/2} = 1.23$$

So the performance improvement is degraded from 2 to 1.23 because cache misses

Example (cont.)

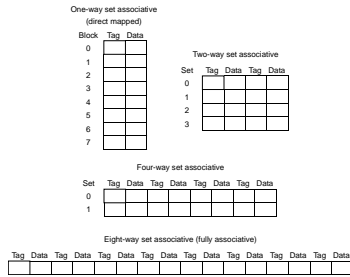
- If we improve both - lower CPI and higher clock
- The lower the CPI the more pronounced is the impact of stall cycles
- The higher the processor rate - larger miss penalty

Improving Performance

- Two ways of improving performance:
 - decreasing the miss ratio
 - decreasing the miss penalty

What happens if we increase block size?

Decreasing miss ratio with associativity



Position of a memory block

- In direct-mapped:
 - (Block number) modulo (Number of cache blocks)
- In set-associative, the set containing the memory block:
 - (Block number) modulo (Number of sets in the cache)

Misses and Associativity in Caches

- Assume three caches, each of four one-word blocks
- One cache is fully associative, a second is two-way set associative, and the third is direct mapped
- Find the number of misses for each cache given the following sequence of addresses: 0, 8, 0, 6, 8
- Direct mapped:
 - Cache block = Block address mod 4
 - 0=0 mod 4, 2=6 mod 4, 0=8 mod 4

Address of memory Block accessed	Hit or miss	Contents of a cache blocks after reference			
		0	1	2	3
0	miss	Mem[0]			
8	miss	Mem[8]			
0	miss	Mem[0]			
6	miss	Mem[0]		Mem[6]	
8	miss	Mem[8]		Mem[6]	

Misses and Associativity in Caches

- The set-associative has two sets (with indices 0 and 1) with two elements per set.
- Set = address mod 2
- 0=0 mod 2, 0=6 mod 2, 0=8 mod 2
- Replace the least recently used block within a set

Address of memory Block accessed	Hit or miss	Contents of a cache blocks after reference			
		Set0	Set0	Set1	Set1
0	miss	Mem[0]			
8	miss	Mem[0]	Mem[8]		
0	hit	Mem[0]	Mem[8]		
6	miss	Mem[0]	Mem[6]		
8	miss	Mem[8]	Mem[6]		

Misses and Associativity in Caches

- The fully associative has four blocks. Three misses
- If we had 8 blocks in cache, there would be no replacements - three misses
- If we had 16 blocks - the three caches would have the same performance
- Cache size and associativity are related in determining cache performance

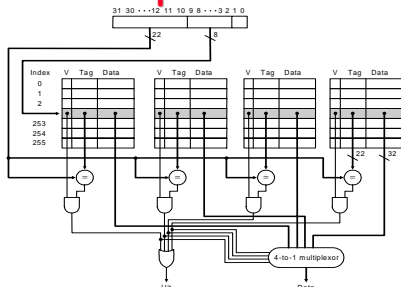
Address of memory Block accessed	Hit or miss	Contents of a cache blocks after reference			
		Block0	Block1	Block2	Block3
0	miss	Mem[0]			
8	miss	Mem[0]	Mem[8]		
0	hit	Mem[0]	Mem[8]		
6	miss	Mem[0]	Mem[8]	Mem[6]	
8	hit	Mem[8]	Mem[8]	Mem[6]	

Performance

- SPEC2000 benchmarks for a 64KB data cache with a 16-word block. Associativity from one to eight way

Associativity	Data miss rate
1	10.3%
2	8.6%
4	8.3%
8	8.1%

An implementation



Tags are searched in parallel
 Each increase by two in associativity doubles the number of blocks per set and halves the number of sets – decreases the size of index by 1 bit and increase the size of tag by 1 bit

How much associativity ?

- The choice among direct-mapped, set-associative mapping will depend:
 - On the cost of a miss versus
 - The cost of implementing associativity

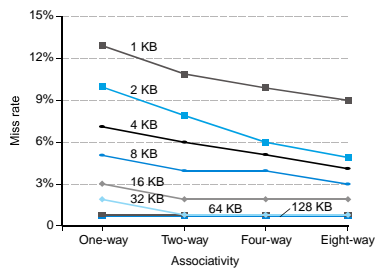
Size of Tags vs. Set Associativity

- Assuming a cache of 4K blocks, a four-word block size and a 32-bit address
- Find the total number of sets and total number of tag bits for caches
 - Direct-mapped
 - Two-way, four-way set associative and full associative
- There are 16 (2^4) bytes per block
- 32 bit address → 32 - 4 = 28 bits to be used for index and tag
- Direct-mapped – same number of sets as blocks
 - $\log_2(4K) = 12$ bits of index, total number of tag bits $(28-12) \times 4K = 64K$ bits
- Each increase by two in associativity doubles the number of blocks per set and halves the number of sets – decreases the size of index by 1 bit and increase the size of tag by 1 bit
- Two-way set-associative – 2K sets and the total number of tag bits: $(28-11)2 \times 2K = 68K$ bits
- Four-way set-associative – 1K sets and the total number of tag bits: $(28-10)4 \times 1K = 72K$ bits
- Full set-associative – one set with 4K blocks, tag is 28 bits and the total number of tag bits: $28 \times 4K = 112K$ bits

Which block to replace?

- Least Recently Used (LRU): The block replaced is the one that has been unused for the longest time.
- Keep track when each element in a set was used relative to other elements in the set.

Performance



Decreasing miss penalty with multilevel caches

- Add a second level cache:
 - often primary cache is on the same chip as the processor
 - use SRAMs to add another cache above primary memory (DRAM)
 - miss penalty goes down if data is in 2nd level cache
- Example:
 - CPI of 1.0 on a 5 GHz machine with a 2% miss rate, 100ns DRAM access
 - Adding 2nd level cache with 5ns access time decreases miss rate to 0.5%. How much faster will the processor be?

Performance

- The miss penalty to main memory is:

$$\frac{100 \text{ ns}}{0.2 \frac{\text{ns}}{\text{Clock cycle}}} = 500 \text{ clock cycles}$$

- The effective CPI with one level of caching is given:
Total CPI = CPI + Memory-stall cycles per instruction
- For one level of caching:
Total CPI = 1 + 2% x 500 = 11
- For two level of caching a miss in primary cache leads to a secondary cache or main memory. The miss penalty for an access to the second-level cache:

$$\frac{5 \text{ ns}}{0.2} = 25 \text{ clock cycles}$$

Performance

- Total CPI = 1 Primary stalls per instr. + Secondary stalls per instr. =
= 1 + 2% x 25 + 0.5% x 500 = 4

The processor with the secondary cache is faster by:

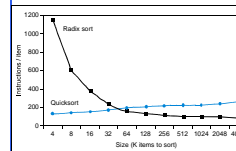
$$11/4 = 2.8$$

Improvements

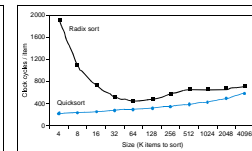
- Using multilevel caches:
 - try and optimize the hit time on the 1st level cache
 - Small and fast
 - try and optimize the miss rate on the 2nd level cache
 - large
- Comparing single to multiple level cache:
 - The primary cache is smaller and uses smaller block size
 - Secondary cache - larger, uses a larger block

Cache Complexities

- Not always easy to understand implications of caches:



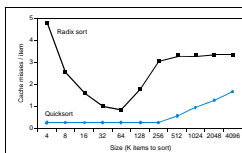
Theoretical behavior of Radix sort vs. Quicksort



Observed behavior of Radix sort vs. Quicksort

Cache Complexities

- Here is why:

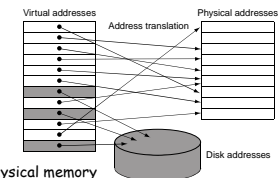


- Memory system performance is often critical factor
 - multilevel caches, pipelined processors, make it harder to predict outcome
 - Compiler optimizations to increase locality sometimes hurt ILP

- Difficult to predict best algorithm: need experimental data

Virtual Memory

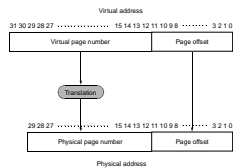
- Main memory can act as a cache for the secondary storage (disk) -
- To allow efficient and safe sharing of memory among multiple programs
- Program to exceed the size of memory



- Advantages:
 - illusion of having more physical memory
 - program relocation
 - Protection - by address translation

Pages: virtual memory blocks

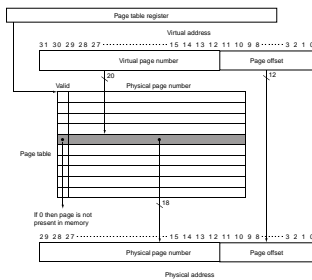
- Page faults: the data is not in memory, retrieve it from disk
 - huge miss penalty, thus pages should be fairly large
 - Typical 4-16KB, new computers 32-64KB, but embedded systems 1KB
 - reducing page faults is important, LRU is worth the price, fully associative
 - can handle the faults in software instead of hardware
 - using write-through is too expensive so we use writeback



Page Tables

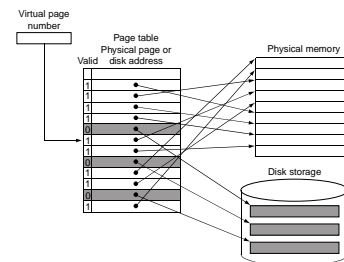
- Pages are located using a Page Table. Each program has its own page table.
 - The page table, together with the program counter and registers, specifies the *state* of a program
 - The *state* or the *process* is saved when the processor is to be used by another program

Page Tables



The page table register indicates the location of the page table in memory

Page Tables



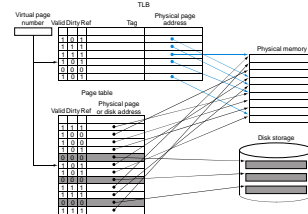
The page table maps each page in virtual memory to either a page in main memory or a page stored on disk, which is the next level of hierarchy

Least Recently Used - LRU

- Implementing a complete accurate LRU is too expensive, since it requires updating a data structure on every memory reference
 - Most operating systems approximate LRU by a *use bit* or *reference bit*
 - Reference bit* is set whenever a page is accessed
 - The operating system periodically clears the *reference bits* and later records them

Making Address Translation Fast

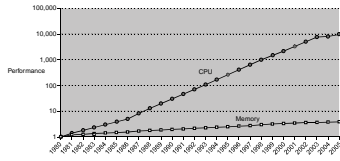
- A cache for address translations: *translation lookaside buffer*



Typical values: 16-512 entries, miss-rate: .01% - 1% miss-penalty: 10 - 100 cycles

Some Issues

- Processor speeds continue to increase very fast
— much faster than either DRAM or disk access times



- Design challenge: dealing with this growing disparity
 - Prefetching? 3rd level caches and more? Memory design?

Summary



- Basics of Caches
- Measuring and Improving Cache Performance
- Framework for Memory Hierarchies