EEC 170
Computer Architecture
Fall 2005

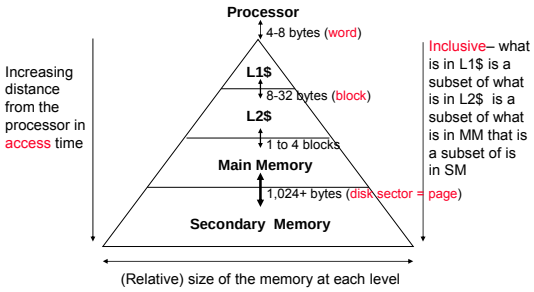
Improving Cache Performance

Administrative

- Problem #6 is posted
  - Last set of homework ☺
  - You should be able to answer each of them in 10-15 min
- Quiz on Wednesday (12/7)
  - Chapter 7: memory hierarchy, caches, virtual memory
  - Lowest quiz grade will be dropped
- Last lecture (12/7)
  - Final review
    - I will email you a sample final by then
  - Problem solving
  - Come prepared with questions to discuss

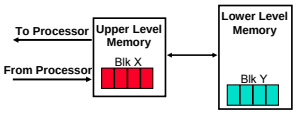
Review: The Memory Hierarchy

- 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



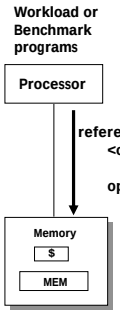
Review: Principle of Locality

- Temporal Locality
  - Keep most recently accessed data items closer to the processor
- Spatial Locality
  - Move blocks consisting of contiguous words to the upper levels



- Hit Time << Miss Penalty
  - Hit: data appears in some block in the upper level (Blk X)
    - Hit Rate: the fraction of accesses found in the upper level
    - Hit Time: RAM access time + Time to determine hit/miss
  - Miss: data needs to be retrieve from a lower level block (Blk Y)
    - Miss Rate = 1 - (Hit Rate)
    - Miss Penalty: Time to replace a block in the upper level with a block from the lower level + Time to deliver this block's word to the processor
    - Miss Types: Compulsory, Conflict, Capacity

The Art of Memory System Design

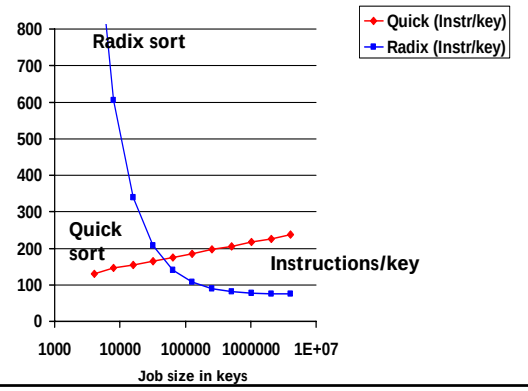


Optimize the memory system organization to minimize the average memory access time for typical workloads

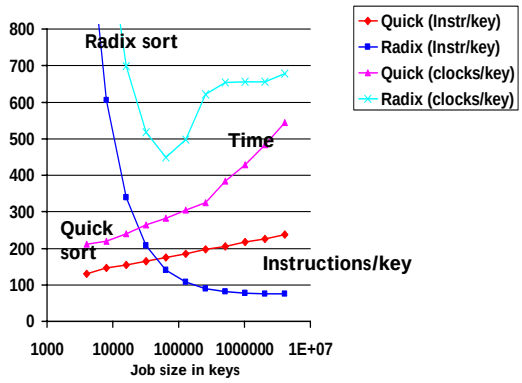
Impact of Memory Hierarchy on Algorithms

- Today CPU time is a function of (ops, cache misses)
- What does this mean to Compilers, Data structures, Algorithms?
  - Quicksort: fastest comparison based sorting algorithm when keys fit in memory. **Complexity O(n.log(n))**
  - Radix sort: also called "linear time" sort. **Complexity O(n)**
    For keys of fixed length and fixed radix a constant number of passes over the data is sufficient independent of the number of keys
- "The Influence of Caches on the Performance of Sorting" by A. LaMarca and R.E. Ladner. Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, January, 1997, 370-379.
  - For Alphastation 250, 32 byte blocks, direct mapped L2 2MB cache, 8 byte keys, from 4000 to 4000000

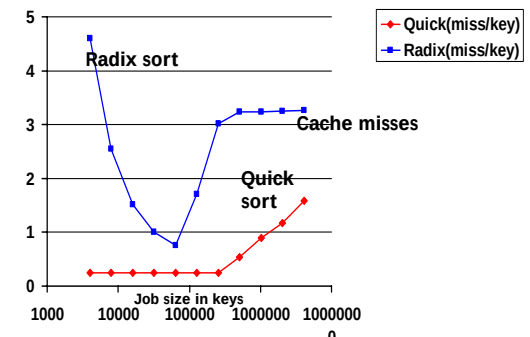
Quicksort vs. Radix as vary number keys: Instructions



Quicksort vs. Radix as vary number keys: Instrs & Time



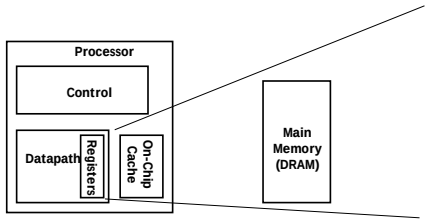
Quicksort vs. Radix as vary number keys: Cache misses



What is proper approach to fast algorithms?

Consider a Simplified Memory System

- Single cache (on-chip) (like your bookshelf)
  - Much smaller than main memory
  - How do we build it? (big/small?)
  - How do we look things up in it? (search/index/etc.?)
  - How do we manage it? (when we run out of space?)



Measuring Cache Performance

- Assuming cache hit costs are included as part of the normal CPU execution cycle, then

$$\text{CPU time} = \text{IC} \times \text{CPI} \times \text{CC}$$

$$= \text{IC} \times \underbrace{(\text{CPI}_{\text{ideal}} + \text{Memory-stall cycles})}_{\text{CPI}_{\text{stall}}} \times \text{CC}$$

- Memory-stall cycles come primarily from cache misses (a sum of read-stalls and write-stalls)

$$\text{Read-stall cycles} = \frac{\text{reads}}{\text{program}} \times \text{read miss rate} \times \text{read miss penalty}$$

$$\text{Write-stall cycles} = \left(\frac{\text{writes}}{\text{program}} \times \text{write miss rate} \times \text{write miss penalty}\right) + \text{write buffer stalls}$$

- For write-through caches, we can simplify this to

$$\text{Memory-stall cycles} = \text{miss rate} \times \text{miss penalty}$$

Impacts of Cache Performance

- Relative cache penalty increases as processor performance improves (faster clock rate and/or lower CPI)
  - The memory system is unlikely to improve as fast as processor cycle time. When calculating $\text{CPI}_{\text{stall}}$, the cache miss penalty is measured in *processor* clock cycles needed to handle a miss
  - The lower the $\text{CPI}_{\text{ideal}}$, the more pronounced the impact of stalls
- A processor with a $\text{CPI}_{\text{ideal}}$ of 2, a 100 cycle miss penalty, 36% load/store instr's, and 2% I$ and 4% D$ miss rates

$$\text{Memory-stall cycles} = 2\% \times 100 + 36\% \times 4\% \times 100 = 3.44$$

$$\text{So} \quad \text{CPI}_{\text{stalls}} = 2 + 3.44 = 5.44$$

- What if the $\text{CPI}_{\text{ideal}}$ is reduced to 1?
- What if the processor clock rate is doubled (doubling the miss penalty)?

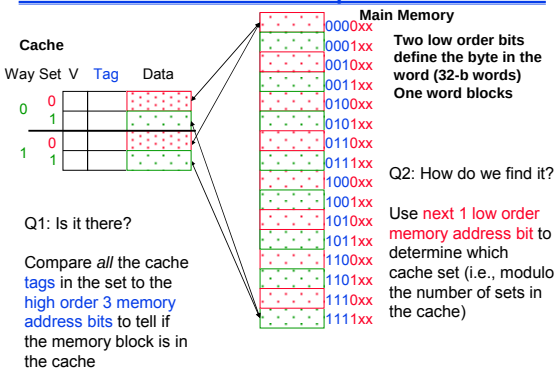
Reducing Cache Miss Rates #1

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)

Set Associative Cache Example



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

Q1: Is it there?

Compare *all* the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache

Q2: How do we find it?

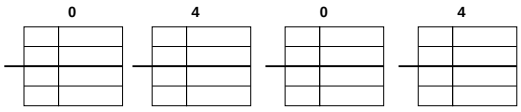
Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache)

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

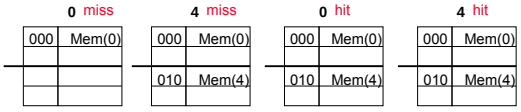


Another Reference String Mapping

- Consider the main memory word reference string

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

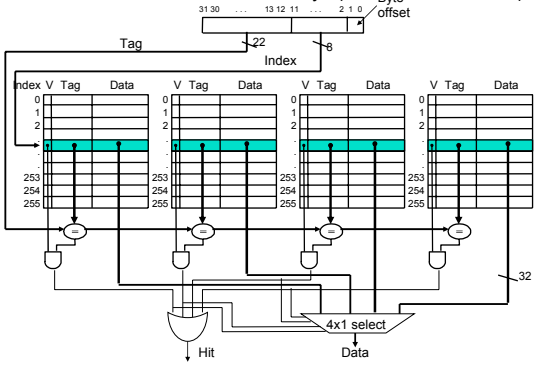
0 4 0 4 0 4 0 4



- 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!

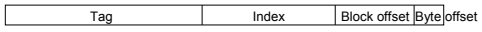
Four-Way Set Associative Cache

- 2^8 = 256 sets each with four ways (each with one block)



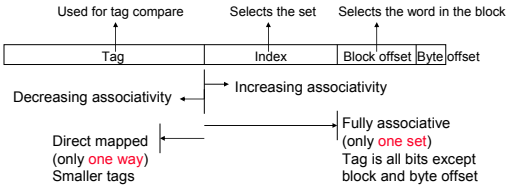
Range of Set Associative Caches

- 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



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

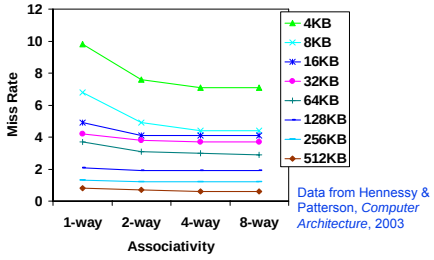


Costs of Set Associative Caches

- 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 – and in some cases even a unified L3 cache
- For our example, CPI_{ideal} of 2, 100 cycle miss penalty (to main memory), 36% load/stores, a 2% (4%) L1 I$ (D$) miss rate, add a UL2$ that has a 25 cycle miss penalty and a 0.5% miss rate

$$CPI_{stalls} = 2 + .02\times25 + .36\times.04\times25 + .005\times100 + .36\times.005\times100 = 3.54$$

(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
- 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

Key Cache Design Parameters

	L1 typical	L2 typical
Total size (blocks)	250 to 2000	4000 to 250,000
Total size (KB)	16 to 64	500 to 8000
Block size (B)	32 to 64	32 to 128
Miss penalty (clocks)	10 to 25	100 to 1000
Miss rates (global for L2)	2% to 5%	0.1% to 2%

Two Machines' Cache Parameters

	Intel P4	AMD Opteron
L1 organization	Split I$ and D$	Split I$ and D$
L1 cache size	8KB for D$, 96KB for trace cache (~I$)	64KB for each of I$ and D$
L1 block size	64 bytes	64 bytes
L1 associativity	4-way set assoc.	2-way set assoc.
L1 replacement	~ LRU	LRU
L1 write policy	write-through	write-back
L2 organization	Unified	Unified
L2 cache size	512KB	1024KB (1MB)
L2 block size	128 bytes	64 bytes
L2 associativity	8-way set assoc.	16-way set assoc.
L2 replacement	~LRU	~LRU
L2 write policy	write-back	write-back

4 Questions for the Memory Hierarchy

- Q1: Where can a block be placed in the upper level? *(Block placement)*
- Q2: How is a block found if it is in the upper level? *(Block identification)*
- Q3: Which block should be replaced on a miss? *(Block replacement)*
- Q4: What happens on a write? *(Write strategy)*

Q1&Q2: Where can a block be placed/found?

	# of sets	Blocks per set
Direct mapped	# of blocks in cache	1
Set associative	(# of blocks in cache)/ associativity	Associativity (typically 2 to 16)
Fully associative	1	# of blocks in cache

	Location method	# of comparisons
Direct mapped	Index	1
Set associative	Index the set; compare set's tags	Degree of associativity
Fully associative	Compare all blocks tags	# of blocks

Q3: Which block should be replaced on a miss?

- Easy for direct mapped – only one choice
- Set associative or fully associative
  - Random
  - LRU (Least Recently Used)
- For a 2-way set associative cache, random replacement has a miss rate about 1.1 times higher than LRU.
- LRU is too costly to implement for high levels of associativity (> 4-way) since tracking the usage information is costly

Q4: What happens on a write?

- *Write-through* – The information is written to both the block in the cache and to the block in the next lower level of the memory hierarchy
  - Write-through is always combined with a write buffer so write waits to lower level memory can be eliminated (as long as the write buffer doesn't fill)
- *Write-back* – The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - Need a dirty bit to keep track of whether the block is clean or dirty
- Pros and cons of each?
  - Write-through: read misses don't result in writes (so are simpler and cheaper)
  - Write-back: repeated writes require only one write to lower level

Improving Cache Performance

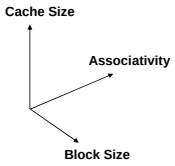
1. Reduce the miss rate
   - bigger cache
   - more flexible placement (increase associativity)
   - larger blocks (16 to 64 bytes typical)
   - victim cache – small buffer holding most recently discarded blocks
2. Reduce the miss penalty
   - smaller blocks
   - use a write buffer to hold dirty blocks being replaced so don't have to wait for the write to complete before reading
   - check write buffer on read miss – may get lucky
   - for large blocks fetch critical word first
   - use multiple cache levels – L2 cache not tied to CPU clock rate
   - faster backing store/improved memory bandwidth
     - wider buses
     - memory interleaving, page mode DRAMs

Improving Cache Performance

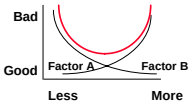
3. Reduce the time to hit in the cache
   - smaller cache
   - direct mapped cache
   - smaller blocks
   - for writes
     - no write allocate – no "hit" on cache, just write to write buffer
     - write allocate – to avoid two cycles (first check for hit, then write) pipeline writes via a delayed write buffer to cache

Summary: The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation

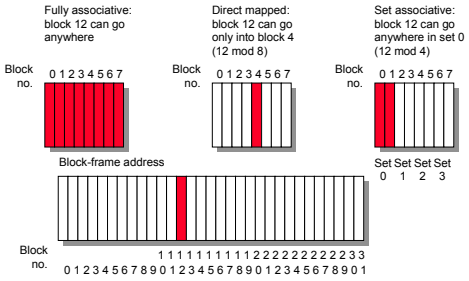


- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins

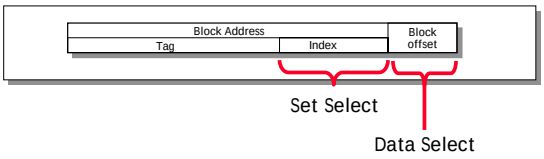


Recap Q1: Where can a block be placed in the upper level?

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets



Recap Q2: How is a block found if it is in the upper level?



- Direct indexing (using index and block offset), tag compares, or combination
- Increasing associativity shrinks index, expands tag

Recap Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Associativity	2 way	2 way	4 way	4 way	8 way	8 way
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Know This!

- Calculate runtime given cache statistics (miss rate, miss penalty, etc.)
- Calculate Average Memory Access Time (AMAT)
- Understand direct mapped, set-associative, fully associative
  - Comparison between them
  - Sources of cache misses
  - Architecture
  - Addressing into them (tag, index, byte)
  - Cache behavior with them

Sample Problem: Impact on Performance

- Suppose a processor executes at
  - Clock Rate = 2 GHz (.5 ns per cycle)
  - Base CPI = 1.1 (assuming 1-cycle cache hits)
  - 50% arith/logic, 30% ld/st, 20% control
- Suppose that 10% of data memory operations (lw, sw) get 50 cycle miss penalty
- Suppose that 1% of instructions get same miss penalty
- What is CPI?
- What is AMAT?

Sample Problem: Impact on Performance

- CPI = Base CPI + average stalls per instruction
  1.1 (cycles/ins) +
  [0.30 (DataMops/ins) x 0.10 (miss/DataMop) x 50(cycle/miss)] +
  [1 (InstMop/ins)x 0.01 (miss/InstMop) x 50 (cycle/miss)]
  = (1.1 + 1.5 + .5) cycle/ins = 3.1
- 64.5% of the time the proc is stalled waiting for memory!
- AMAT=(1/1.3)x[1+0.01x50]+(0.3/1.3)x[1+0.1x50]=2.54

Sample Problem: Cache Performance

- Processor:
  - CPI = 2
  - Icache miss rate = 2%, miss penalty = 100 cycles
  - Dcache miss rate = 4%, miss penalty = 100 cycles
- Using SPECint2000 load/store percentage of 36%:
  - Speedup from this processor to one that never missed?
  - What if CPI = 1?
  - What if we doubled clock rate of computer without changing memory speed (CPI = 2)?

Calculating Cache Performance, CPI=2

- I miss cycles = I * 2% * 100 = 2.00 * I
- D miss cycles = I * 4% * 36% * 100 = 1.44 * I
- So memory stalls/instr = 2.00 + 1.44 = 3.44

- CPU time with stalls = $I * CPI_{stall} * clkcycle$
- CPU time, no stalls $I * CPI_{perf} * clkcycle$

- $CPI_{stall} = 2 + 3.44$; $CPI_{perf} = 2$
- $CPI_{stall} / CPI_{perf} = 5.44 / 2 = 2.72$

Calculating Cache Performance, CPI=1

- I miss cycles = I * 2% * 100 = 2.00 * I
- D miss cycles = I * 4% * 36% * 100 = 1.44 * I
- So memory stalls/instr = 2.00 + 1.44 = 3.44

- CPU time with stalls = $I * CPI_{stall} * clkcycle$
- CPU time, no stalls $I * CPI_{perf} * clkcycle$

- $CPI_{stall} = 1 + 3.44$; $CPI_{perf} = 1$
- $CPI_{stall} / CPI_{perf} = 4.44 / 4 = 4.44$

Calculating Cache Performance, 2x

- I miss cycles = I * 2% * 200 = 4.00 * I
- D miss cycles = I * 4% * 36% * 200 = 2.88 * I
- So memory stalls/instr = 4.00 + 2.88 = 6.88

- CPU time (slow clk) = $I * CPI_{slow} * clkcycle$
- CPU time (fast clk) $I * CPI_{fast} * clkcycle/2$

- $CPI_{fast} = 2 + 6.88$; $CPI_{slow} = 5.44$
- Speedup = 5.44 / (8.88 * 0.5) = 1.23
- Ideal machine is 2x faster