CS 61C: Great Ideas in Computer Architecture (Machine Structures)
More Cache: Set Associativity

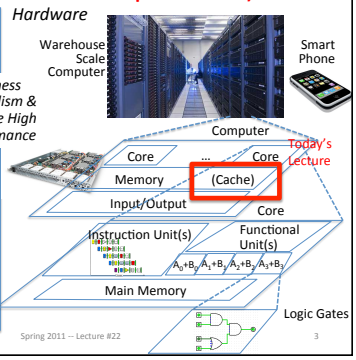
Instructors:
Randy H. Katz
David A. Patterson
Guest Lecture: Krste Asanovic
http://inst.eecs.Berkeley.edu/~cs61c/fa10

New-School Machine Structures (It's a bit more complicated!)

Software | *Hardware*

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time

Harness Parallelism & Achieve High Performance



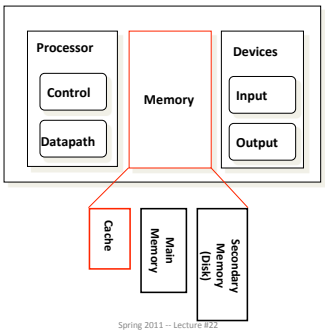
Review

- Big Ideas of Instruction-Level Parallelism
- Pipelining, Hazards, and Stalls
- Forwarding, Speculation to overcome Hazards
- Multiple issue to increase performance
  – IPC instead of CPI
- Dynamic Execution: Superscalar in-order issue, branch prediction, register renaming, out-of-order execution, in-order commit
  – "unroll loops in HW", hide cache misses

Agenda

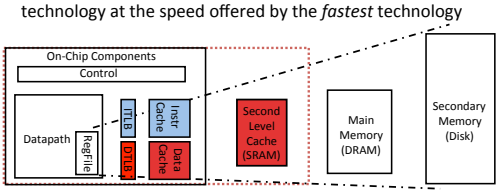
- Cache Memory Recap
- Administrivia
- Set-Associative Caches
- Technology Break
- AMAT and Multilevel Cache Review

Recap: Components of a Computer



Recap: 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



Speed (%cycles):	½'s	1's	10's	100's	10,000's
Size (bytes):	100's	10K's	M's	G's	T's
Cost:	highest				lowest

Recap: Cache Performance and Average Memory Access Time (AMAT)

- CPU time = IC × CPI × CC
  = IC × (CPI_{ideal} + Memory-stall cycles) × CC

  CPI_{stall}

  Memory-stall cycles = Read-stall cycles + Write-stall cycles

  Read-stall cycles = reads/program × read miss rate × read miss penalty

  Write-stall cycles = (writes/program × write miss rate × write miss penalty) + *write buffer stalls*
- AMAT is the average time to access memory considering both hits and misses

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

Improving Cache Performance

- Reduce the time to hit in the cache
  - E.g., Smaller cache, direct-mapped cache, special tricks for handling writes
- *Reduce the miss rate*
  - E.g., Bigger cache, larger blocks
  - *More flexible placement (increase associativity)*
- Reduce the miss penalty
  - E.g., Smaller blocks or critical word first in large blocks, special tricks for handling writes, faster/higher bandwidth memories
  - Use multiple cache levels

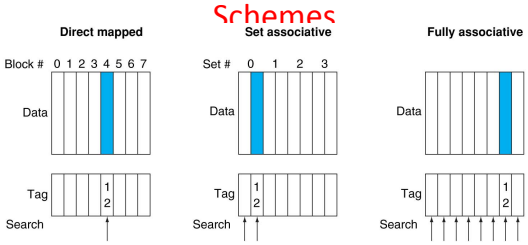
Sources of Cache Misses: The 3Cs

- Compulsory (cold start or process migration, 1st reference):
  - First access to block impossible to avoid; small effect for long running programs
  - 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 (may increase access time)

Reducing Cache Misses

- Allow more flexible block placement
- *Direct mapped $*: memory block maps to exactly one cache block
- *Fully associative $*: allow a memory block to be mapped to any cache block
- Compromise: divide $ into sets, each of which consists of n "ways" (*n-way set associative*) to place memory block
  - Memory block maps to unique set determined by index field and is placed in any of the n-ways of that set
  - Calculation: (block address) modulo (# sets in the cache)

Alternative Block Placement Schemes



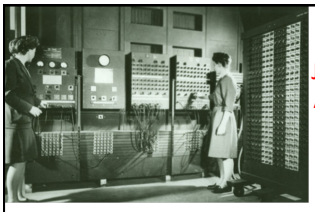
- DM placement: mem block 12 in 8 block cache: only one cache block where mem block 12 can be found—(12 modulo 8) = 4
- SA placement: four sets x 2-ways (8 cache blocks), memory block 12 in set (12 mod 4) = 0; either element of the set
- FA placement: mem block 12 can appear in any cache blocks

Agenda

- Cache Memory Recap
- Administrivia
- Set-Associative Caches
- Technology Break
- AMAT and Multilevel Cache Review

Administrivia

- Project 4, Part 2 due Sunday 4/17
  - Design a 16-bit pipelined computer in Logisim
- Extra Credit due 4/24 – Fastest Matrix Multiply
- Final Review: Mon 5/2 5 – 8PM, 2050 VLSB
- Final Exam: Mon 5/9 11:30-2:30PM 100 Haas Pavilion



61c in the News: Jean Bartik, dies age 86 A Pioneer Programmer

- Last survivor of group of women programmers of ENIAC, the first all-electronic digital calculator
- Used for calculating artillery tables, but completed in 1946 too late for WWII

"When the Eniac was shown off at the University of Pennsylvania in February 1946, it generated headlines in newspapers across the country. But the attention was all on the men and the machine. The women were not even introduced at the event."

[NY Times, 4/7/2011]

Getting to Know Your Prof

- Missing the ales I grew up with in England, I learnt how to make beer.
- Start from grains, about 5-10 gallons/batch
- Brewed over 50 batches so far

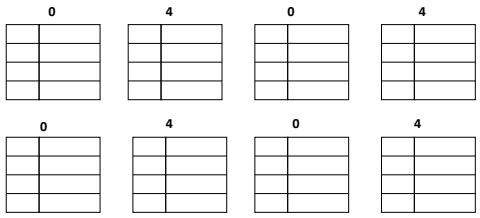
Need thirsty friends to help consume experiments!

Example: 4-Word Direct-Mapped $ Worst-Case Reference String

- 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

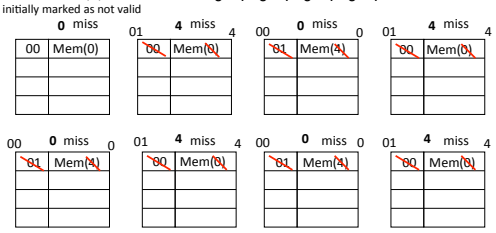


Example: 4-Word Direct-Mapped $ Worst-Case Reference String

- Consider the main memory word reference string

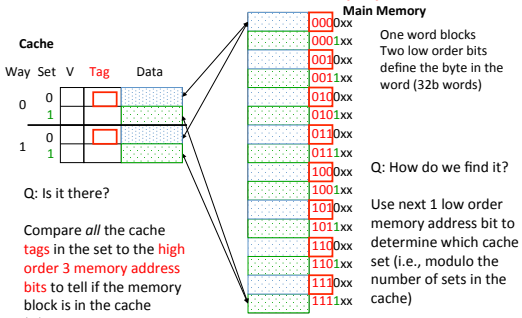
Start with an empty cache - all blocks initially marked as not valid

0 4 0 4 0 4 0 4



- 8 requests, 8 misses
- Ping pong effect due to conflict misses - two memory locations that map into the same cache block

Example: 2-Way Set Associative $
(4 words = 2 sets x 2 ways per set)



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

Q: 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

Q: How do we find it?

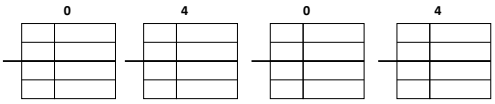
Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache)

Example: 4 Word 2-Way SA $ Same Reference String

- 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

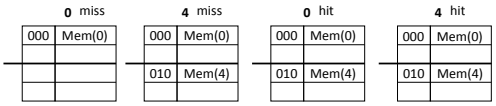


Example: 4-Word 2-Way SA $ Same Reference String

- Consider the main memory word reference string

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

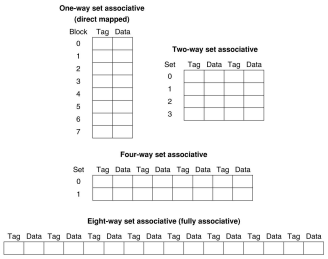
0 4 0 4 0 4 0 4



- 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!

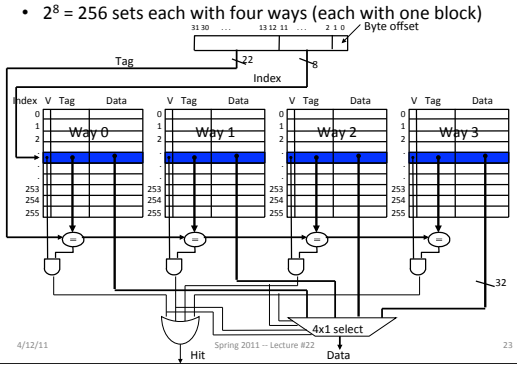
Example: Eight-Block Cache with Different Organizations



Total size of $ in blocks is equal to *number of sets* x *associativity*. For fixed $ size, increasing associativity decreases number of sets while increasing number of elements per set. With eight blocks, an 8-way set-associative $ is same as a fully associative $.

Four-Way Set-Associative Cache

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



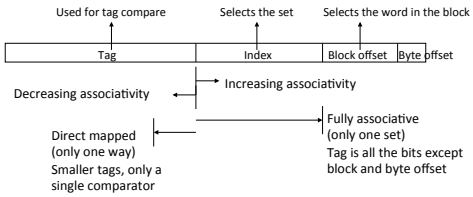
Range of Set-Associative Caches

- 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

Tag	Index	Block offset	Byte offset

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

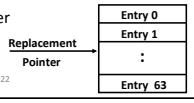


Costs of Set-Associative Caches

- When miss occurs, which way's block selected for replacement?
  - Least Recently Used (LRU): one that has been unused the longest
    - Must track when each way's block was used relative to other blocks in the set
    - For 2-way SA $, one bit per set → set to 1 when a block is referenced; reset the other way's bit (i.e., "last used")
- 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). DM $: block is available before the Hit/Miss decision
    - In Set-Associative, not possible to just assume a hit and continue and recover later if it was a miss

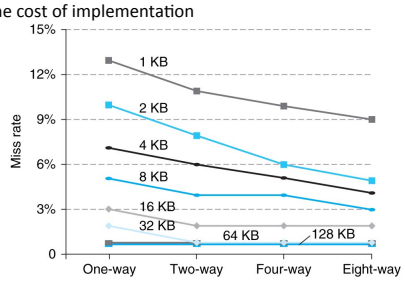
Cache Block Replacement Policies

- Random Replacement
  - Hardware randomly selects a cache item and throw it out
- Least Recently Used
  - Hardware keeps track of access history
  - Replace the entry that has not been used for the longest time
  - For 2-way set-associative cache, need one bit for LRU replacement
- Example of a Simple "Pseudo" LRU Implementation
  - Assume 64 Fully Associative entries
  - Hardware replacement pointer points to one cache entry
  - Whenever access is made to the entry the pointer points to:
    - Move the pointer to the next entry
  - Otherwise: do not move the pointer



Benefits of Set-Associative Caches

- Choice of DM $ or SA $ depends on the cost of a miss versus the cost of implementation

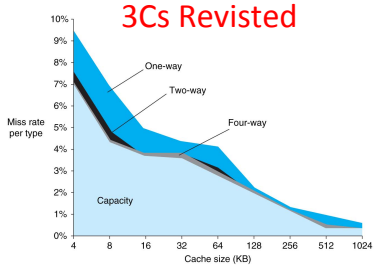


- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)

How to Calculate 3C's using Cache Simulator

1. *Compulsory*: set cache size to infinity and fully associative, and count number of misses
2. *Capacity*: Chance cache size from infinity, usually in powers of 2, and count misses for each reduction in size
   - 16 MB, 8 MB, 4 MB, … 128 KB, 64 KB, 16 KB
3. *Conflict*: Change from fully associative to n-way set associative while counting misses
   - Fully associative, 16-way, 8-way, 4-way, 2-way, 1-way

3Cs Revisted



- Three sources of misses (SPEC2000 integer and floating-point benchmarks)
  - Compulsory misses 0.006%; not visible
  - Capacity misses, function of cache size
  - Conflict portion depends on associativity and cache size

Reduce AMAT

- Use multiple levels of cache
- As technology advances, more room on IC die for larger L1$ or for additional levels of cache (e.g., L2$ and L3$)
- Normally the higher cache levels are unified, holding both instructions and data

AMAT Revisited

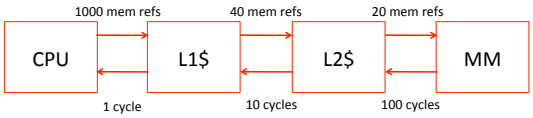
- For a 2nd-level cache, L2 Equations:
  AMAT = Hit Time_{L1} + Miss Rate_{L1} x Miss Penalty_{L1}
  Miss Penalty_{L1} = Hit Time_{L2} + Miss Rate_{L2} x Miss Penalty_{L2}
  AMAT = Hit Time_{L1} + Miss Rate_{L1} x (Hit Time_{L2} + Miss Rate_{L2} × Miss Penalty_{L2})
- Definitions:
  - *Local miss rate*: misses in this $ divided by the total number of memory accesses to this $ (Miss rate_{L2})
  - *Global miss rate*: misses in this $ divided by the total number of memory accesses generated by the CPU (Miss Rate_{L1} x Miss Rate_{L2})
  - Global miss rate is what matters to overall performance
  - Local miss rate is factor in evaluating the effectiveness of L2 cache

$\text{CPI}_{\text{stalls}}$ Calculation

- Assume
  - $\text{CPI}_{\text{ideal}}$ of 2
  - 100 cycle miss penalty to main memory
  - 25 cycle miss penalty to Unified L2$
  - 36% of instructions are load/stores
  - 2% L1 I$ miss rate; 4% L1 D$ miss rate
  - 0.5% U(nified)L2$ miss rate
- $\text{CPI}_{\text{stalls}}$ = 2 + 1x.02×25 + .36×.04×25 (L1)
  + 1x.005×100 + .36×.005×100 (L2)
  = 3.54 (vs. 5.44 with no L2$)

IFetch
Ld/St

Memory Hierarchy with Two Cache Levels



- For every 1000 CPU to memory references
  - 40 will miss in L1$; what is the miss rate?
  - 20 will miss in L2$; what is the miss rate?
  - Global vs. local miss rate?
- If 1.5 mem refs per instruction, how do we normalize these numbers to # instructions? Ave. Mem Stalls/Instruction

AMAT Calculations Local vs. Global Miss Rates

Example:
- For 1000 memory refs:
  40 misses in L1$ (miss rate 4%)
  20 misses in L2$ (miss rate 2%)
- L1$ hits 1 cycle,
  L2$ hits in 10 cycles
  Miss to MM costs 100 cycles
- 1.5 memory references per instruction (i.e., 50% ld/st)
  1000 mem refs = 667 instrs OR
  1000 instrs = 1500 mem refs

Ask:
Local miss rate
AMAT
Stall cycles per instruction with and without L2$

With L2$
- Local miss rate =
- AMAT =
- Ave Mem Stalls/Ref =
- Ave Mem Stalls/Instr=

Without L2$
- AMAT=
- Ave Mem Stalls/Ref=
- Ave Mem Stalls/Instr=

Assume ideal CPI=1.0, performance improvement =

AMAT Calculations Local vs. Global Miss Rates

Example:
- For 1000 memory refs:
  40 misses in L1$ (miss rate 4%)
  20 misses in L2$ (miss rate 2%)
- L1$ hits 1 cycle,
  L2$ hits in 10 cycles
  Miss to MM costs 100 cycles
- 1.5 memory references per instruction (i.e., 50% ld/st)
  1000 mem refs = 667 instrs OR
  1000 instrs = 1500 mem refs

Ask:
Local miss rate
AMAT
Stall cycles per instruction with and without L2$

With L2$
- Local miss rate = 50% (20/40)
- AMAT = 1+4%x(10+50%x100)=3.4
- Ave Mem Stalls/Ref = (3.4-1.0)=2.4
- Ave Mem Stalls/Instr=2.4x1.5=3.6

Without L2$
- AMAT=1+4%x100=5
- Ave Mem Stalls/Ref=(5-1.0)=4
- Ave Mem Stalls/Instr=4x1.5=6

Assume ideal CPI=1.0, performance improvement = (6+1)/(3.6+1)=52%

CPI/Miss Rates/DRAM Access SpecInt2006

Name	CPI	L1 D cache misses/1000 instr	L2 D cache misses/1000 instr	DRAM accesses/1000 instr
perl	0.75	3.5	1.1	1.3
bzip2	0.85	11.0	5.8	2.5
gcc	1.72	24.3	13.4	14.8
mcf	10.00	106.8	88.0	88.5
go	1.09	4.5	1.4	1.7
hmmer	0.80	4.4	2.5	0.6
sjeng	0.96	1.9	0.6	0.8
libquantum	1.61	33.0	33.1	47.7
h264avc	0.80	8.8	1.6	0.2
omnetpp	2.94	30.9	27.7	29.8
astar	1.79	16.3	9.2	8.2
xalancbmk	2.70	38.0	15.8	11.4
Median	1.35	13.6	7.5	5.4

Design Considerations

- Different design considerations for L1$ and L2$
  - L1$ focuses on fast access: minimize hit time to achieve shorter clock cycle, e.g., smaller $
  - L2$, L3$ focus on low miss rate: reduce penalty of long main memory access times: e.g., Larger $ with larger block sizes/ higher levels of associativity
- Miss penalty of L1$ is significantly reduced by presence of L2$, so can be smaller/faster even with higher miss rate
- For the L2$, fast hit time is less important than low miss rate
  - L2$ hit time determines L1$'s miss penalty
  - L2$ local miss rate >> than the global miss rate

Improving Cache Performance

- Reduce the time to hit in the cache
  - E.g., Smaller cache, direct-mapped cache, special tricks for handling writes
- *Reduce the miss rate (in L2$, L3$)*
  - *E.g., Bigger cache, larger blocks*
  - *More flexible placement (increase associativity)*
- *Reduce the miss penalty (in L1$)*
  - *E.g., Smaller blocks or critical word first in large blocks, special tricks for handling for writes, faster/ higher bandwidth memories*
  - *Use multiple cache levels*

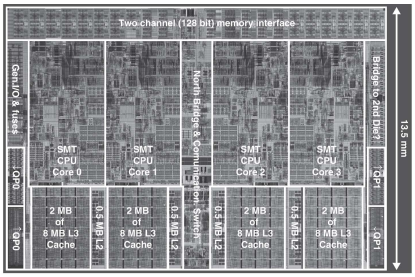
Sources of Cache Misses: 3Cs for L2$, L3$

- Compulsory (cold start or process migration, 1st reference):
  - First access to block impossible to avoid; small effect for long running programs
  - 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 (may increase access time)*

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; 256KB (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

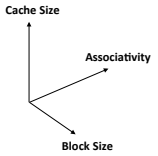
Intel Nehalem Die Photo

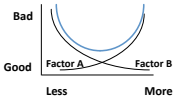


- 4 cores, 32KB I$/32-KB D$, 512KB L2$
- Share one 8-MB L3$

Cache Design Space

- Several interacting dimensions
  - Cache size
  - Block size
  - Associativity
  - Replacement policy
  - Write-through vs. write-back
  - Write allocation
- Optimal choice is a compromise
  - Depends on access characteristics
    - Workload
    - Use (I-cache, D-cache)
  - Depends on technology / cost
- Simplicity often wins





Summary

- Name of the Game: Reduce Cache Misses
  - 2 memory blocks mapping to same block knock each other out as program bounces from 1 memory location to next
- One way to do it: set-associativity
  - Memory block maps into more than 1 cache block
  - N-way: n possible places in cache to hold a memory block
- N-way Cache of 2^{N+M} blocks: 2^N ways x 2^M sets
- Multi-level caches
  - Optimize first level to be fast!
  - Optimize 2nd and 3rd levels to minimize the memory access penalty