CS 61C:
Great Ideas in Computer Architecture
More Cache: Set Associativity

Instructors:
Krste Asanovic, Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/fa12

Review

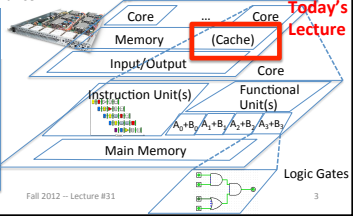
- Big Ideas of Instruction-Level Parallelism
- Pipelining, Hazards, and Stalls
- Forwarding, Speculation to overcome Hazards
- Multiple issue to increase performance
  - IPC instead of CPI
- Dynamic Execution: Superscalar in-order issue, branch prediction, register renaming, out-of-order execution, in-order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

You Are Here!

Software
- 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
- Programming Languages

Harness Parallelism & Achieve High Performance

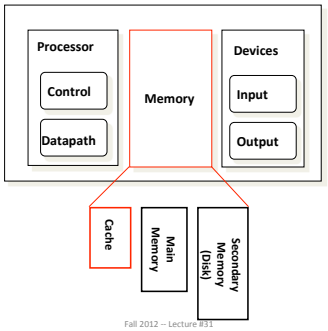
Hardware



Agenda

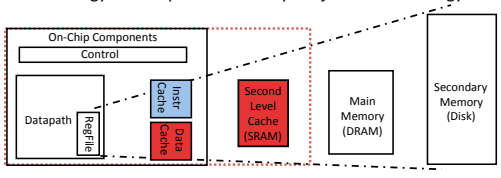
- Cache Memory Recap
- Administrivia
- Set-Associative Caches
- AMAT and Multilevel Cache Review
- Nehalem Memory Hierarchy

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

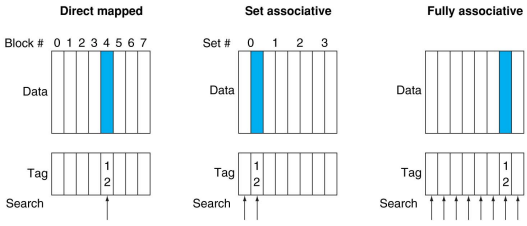
Sources of Cache Misses: The 3Cs

- 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)

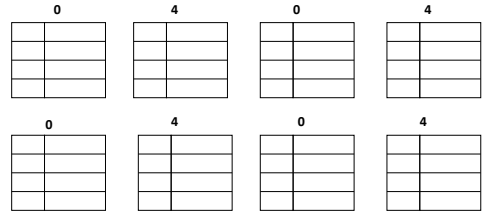
Alternative Block Placement Schemes



- 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

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

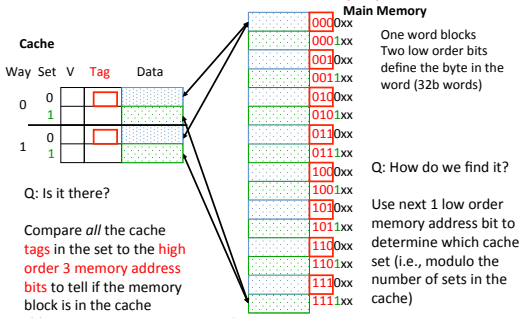
- Consider the main memory word reference string

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?

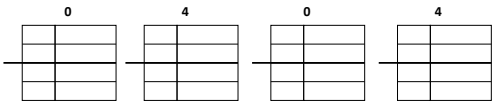
Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache)

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

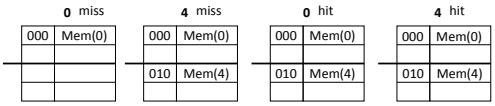


Example: 4-Word 2-Way SA $ Same Reference String

- Consider the main memory word reference string

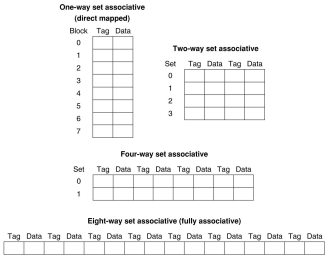
Start with an empty cache - all blocks initially marked as not valid

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!

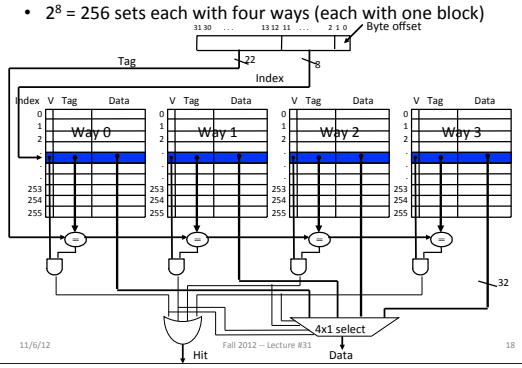
Example: Eight-Block Cache with Different Organizations



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)



Flashcard Quiz: For fixed capacity and fixed block size, how does increasing associativity effect AMAT?

Increases hit time, decreases miss rate

Decreases hit time, decreases miss rate

Increases hit time, increases miss rate

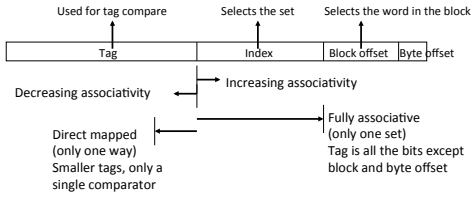
Decreases hit time, decreases miss rate

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

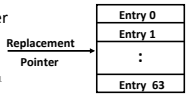


Costs of Set-Associative Caches

- When 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

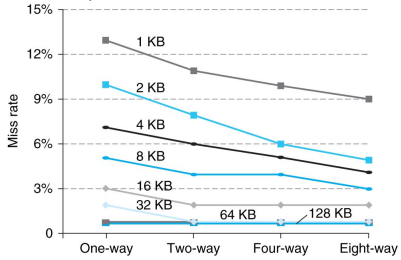
Cache Block Replacement Policies

- 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)

Administrivia

Final Exam Monday Dec 10: 11:30-2:30

220/230/242 Hearst Gym

CS61c in the News: End of MIPS?

ARM, Imagination divvy up MIPS

Junko Yoshida, EE Times

11/6/2012 9:41 AM EST

NEW YORK -- MIPS Technologies, which had been on the block for almost a year, finally found buyers in a complicated deal involving Imagination Technologies and ARM.

Imagination said Tuesday (Nov. 6) it has agreed to buy MIPS' operating business for $60 million. Under terms of the deal, the U.K. graphics IP vendor will gain 160 engineers and 82 MIPS patents.

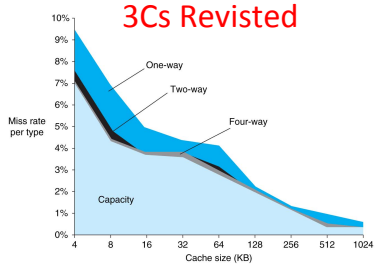
The move is viewed as a way for Imagination to beef up its CPU core expertise while defending its graphics lead. It would also position Imagination to competing against ARM, which has been pursuing its integrated CPU-GPU solution strategy.

Separately, ARM said it will lead a consortium buying the rights to the MIPS portfolio of 498 patents. The consortium, called Bridge Crossing LLC, will pay $350 million in cash to purchase the rights to the portfolio, of which ARM will contribute $167.5 million.

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*: Change 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

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 for writes, faster/ higher bandwidth memories*
  - *Use multiple cache levels*

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

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

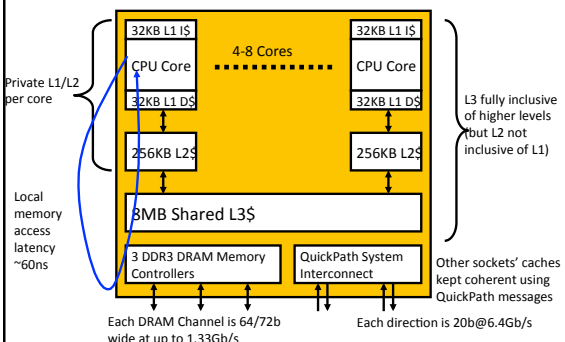
Flashcard Quiz: In a machine with two levels of cache, what effect does increasing L1 capacity have on L2*?

Decreases L2 capacity misses
Increases L2 local miss rate
Decreases L2 compulsory misses
Decreases L2 global miss rate

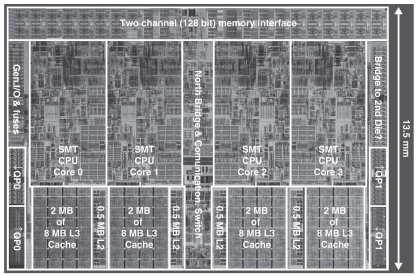
Two Machines' Cache Parameters

	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; 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

Nehalem Memory Hierarchy Overview



Intel Nehalem Die Photo



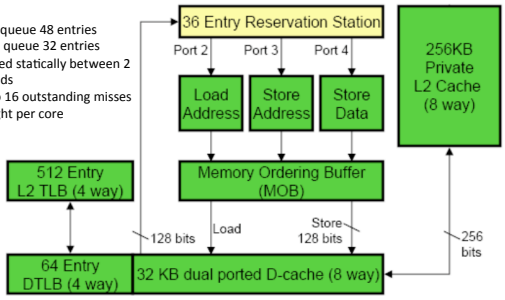
- 4 cores, 32KB I$/32-KB D$, 512KB L2$
- Share one 8-MB L3$

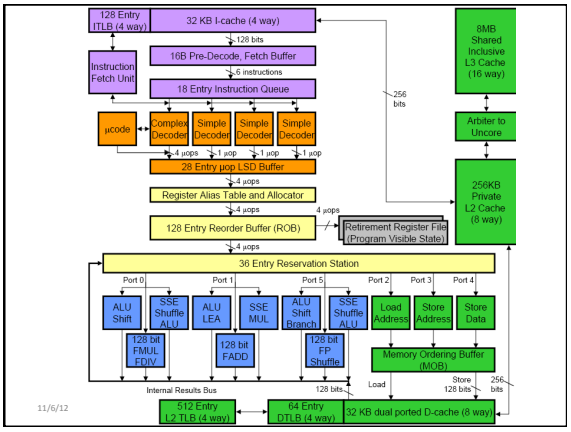
Cache Hierarchy Latencies

- L1 I & D 32KB 8-way, latency 4 cycles, 64B blocks
- L2 256 KB 8-way, latency <12 cycles
- L3 8 MB, 16-way, latency 30-40 cycles
- DRAM, latency ~180-200 cycles

Core's Private Memory System

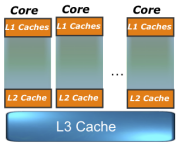
Load queue 48 entries
Store queue 32 entries
Divided statically between 2 threads
Up to 16 outstanding misses in flight per core





Intel® Smart Cache – 3rd Level Cache

- Shared across all cores
- Size depends on # of cores
  - Quad-core: Up to 8MB (16-ways)
  - ***Scalability:***
    - Built to vary size with varied core counts
    - Built to easily increase L3 size in future parts
- Perceived latency depends on frequency ratio between core & uncore
- Inclusive cache policy for best ***performance***
  - Address residing in L1/L2 ***must*** be present in 3rd level cache



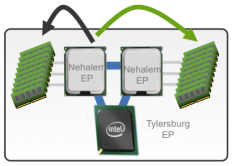
Why Inclusive?

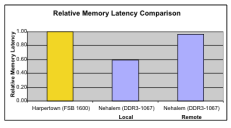
- Inclusive cache provides benefit of an on-die snoop filter
- Core Valid Bits
  - 1 bit per core per cache line
    - If line may be in a core, set core valid bit
    - Snoop only needed if line is in L3 and ***core valid*** bit is set
    - Guaranteed that line is not modified if multiple bits set
- ***Scalability***
  - Addition of cores/sockets does not increase snoop traffic seen by cores
- ***Latency***
  - Minimize effective cache latency by eliminating cross-core snoops in the common case
  - Minimize snoop response time for cross-socket cases

Non-Uniform Memory Access (NUMA)

- FSB architecture
  - All memory in one location
- Starting with Intel® Core™ microarchitecture (Nehalem)
  - Memory located in multiple places
- Latency to memory dependent on location
- Local memory has highest BW, lowest latency
- Remote Memory still very fast

Ensure software is NUMA-optimized for best performance





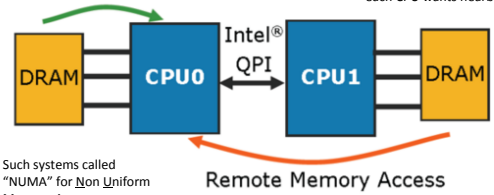
Intel® Core™ microarchitecture (Nehalem-EP)
Intel® Next Generation Server Processor Technology (Tylersburg-EP)

All Sockets can Access all Data

How ensure that get data allocated to local DRAM?

Lunix doesn't allocate pages to physical memory after malloc until first access to page. Be sure to touch what each CPU wants nearby

~60ns
Local Memory Access

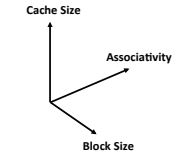


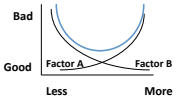
Remote Memory Access
~100ns

Such systems called "NUMA" for Non Uniform Memory Access: some addresses are slower than others

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