CS 61C: Great Ideas in Computer Architecture (Machine Structures)
Multi-Level Caches

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

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
    - Not possible to just assume a hit and continue and recover later if it was a miss

Agenda

- Multi-level Caches
- Administrivia
- Technology Break
- Writes Revisted

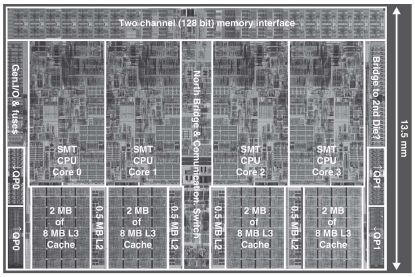
Agenda

- Muilt-Level Caches
- Administrivia
- Technology Break
- Writes Revisted

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

Intel Nehalem Die Photo



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

AMAT Revisited

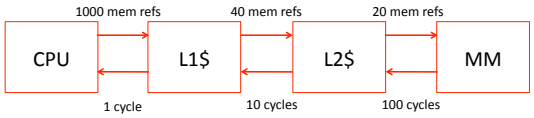
- For a 2nd-level cache, L2$ Equations:
  AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x Miss Penalty$_{L1}$
  Miss Penalty$_{L1}$ = Hit Time$_{L2}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$
  AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x (Hit Time$_{L2}$ + Miss Rate$_{L2}$ × Miss Penalty$_{L2}$)
- Definitions:
  - *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 effectiveness of L2$

CPI$_{stalls}$ Calculation

- Assume
  - CPI$_{ideal}$ 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
- CPI$_{stalls}$ = 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

Two Cache Levels: Global vs. Local Cache Misses



- 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 ?)
- 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 ?)
- 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 $ with smaller block sizes
  - 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$, so can be smaller/faster even with higher miss rate
- For L2$, 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, smaller blocks, special tricks for handling for 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; 256MB (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

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; 256MB (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

Agenda

- Multi-Level Caches
- Administrivia
- Technology Break
- Writes Revisited

Administrivia

- Project 3: TLP+DLP+Cache Opt (Due 11/13)
- Project 4: Single Cycle Processor in Logicsim
  - Due Part 2 due Saturday 11/27
  - Face-to-Face in lab 12/2
- EC: Fastest Project 3 (due 11/29)
- Final Review: Mon Dec 6, 3 hrs, afternoon (TBD)
- Final: Mon Dec 13 8AM-11AM (TBD)
  - Like midterm: T/F, M/C, short answers
  - Whole Course: readings, lectures, projects, labs, hw
  - Emphasize 2nd half of 61C + midterm mistakes

Agenda

- Multi-Level Caches
- Administrivia
- Technology Break
- Writes Revisited

Agenda

- Multi-level Caches
- Administrivia
- Technology Break
- Writes Revisited

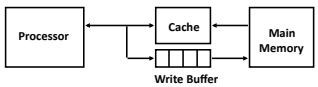
Cache Write Policy

- $ read is much easier to handle than $ write
  - I$ is much easier to design than D$
- Cache write
  - How do we keep cached data and memory consistent?
- Two options
  - Write Back: write to cache only. Write the cache block to memory when that cache block is being replaced on a cache miss
    - Need a "dirty bit" for each cache block
    - Greatly reduce the memory bandwidth requirement
    - Control can be complex
  - Write Through: write to cache and memory at the same time
    - Isn't memory too slow for this?
    - Solution: Write Buffer

Write Through vs. Write Back

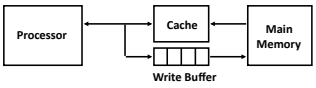
		HIT	MISS
Write Thru	READ	CPU reads cache	CPU detects miss, stalls Cache selects replacement block New block loaded from MM Requested word sent to CPU CPU resumes operation
	WRITE	CPU writes cache CPU writes MM and stalls until write completes	CPU detects miss CPU writes MM (cache also if write allocate) stalls until write completes
		HIT	**MISS**
Write Back	READ	CPU reads cache	CPU detects miss, stalls Cache selects replacement block New block loaded from MM Word sent to CPU CPU resumes operation
	WRITE	CPU writes cache	CPU detects miss, stalls Cache selects replacement block Old block evicted from cache New block loaded from MM (write allocate) CPU resumes operation

Write Buffer for Write Through

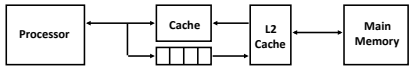


- Write Buffer: between $ and main memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a first-in first-out queue
  - Typical number of entries: 4
  - Works fine if store frequency (wrt. time) << 1 / MM write cycle
- Memory system designer's nightmare
  - Store frequency (wrt. time) > 1 / MM write cycle
  - Write buffer saturation
- Implications of write buffer for multiprocessor caches?

Write Buffer Saturation

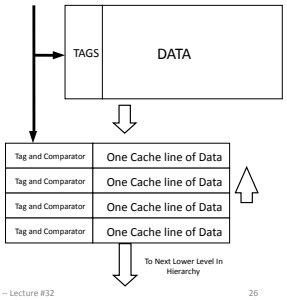


- Store frequency (wrt. time) > 1 / MM write cycle
  - If this condition persists for too long (CPU cycle time too fast and/or too many store instructions in a row)
    - Store buffer will overflow no matter how big you make it
    - CPU Cycle Time << MM Write Cycle Time
- Solutions for write buffer saturation
  - Use a write back cache
  - Install a second level (L2) cache

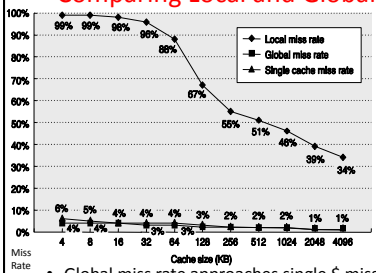


Victim Cache

- How to combine fast hit time of direct mapped $ yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Kind of "cheap and dirty" associativity
- Studies showed that a 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache

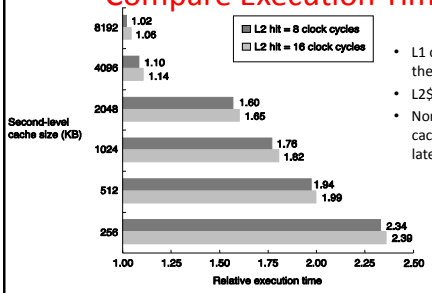


Comparing Local and Global Miss Rates



- L1$: split 64K I$ + 64K D$, 2-way SA
- L2$: 4K to 4M
- Global miss rate approaches single $ miss rate provided L2$ is much larger than the L1$ (*but runs at the speed of the smaller L1$*!)
- Global miss rate more important than local miss rate

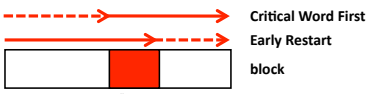
Compare Execution Times



- L1 configuration as in the last slide
- L2$ 256K-8M, 2-way
- Normalized to 8M cache with 1-cycle latency
- Performance is not sensitive to L2 latency
- Larger cache size makes a big difference

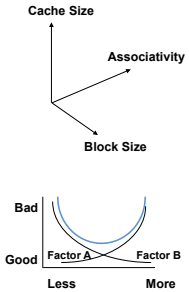
Early Restart and Critical Word First

- Don't wait for full block to be loaded before restarting CPU
  - Early restart—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Critical Word First—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called wrapped fetch and requested word first
- Generally useful only in large blocks (relative to bandwidth)
- Good spatial locality may reduce the benefits of early restart, as the next sequential word may be needed anyway



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, TLB)
  - Depends on technology / cost
- Simplicity often wins



Summary

- Multi-level caches
  - Optimize first level to be fast!
    - I$/D$ hit times affect the basic processor clock
    - Implies relatively small and simple $ organization
  - Optimize second and third levels to minimize the memory access penalty
    - Larger, more complex organization but slower ok
- Optimizing the write case
  - Write Buffers and Victim Caches are some of the tricks