Cache Performance

- CPUtime = Instruction count x CPI x Clock cycle time
- CPIexecution = CPI with ideal memory
- CPI = CPIexecution + Mem Stall cycles per instruction
- Mem Stall cycles per instruction =

Mem accesses per instruction x Miss rate x Miss penalty

- CPUtime = Instruction Count x (CPIexecution + Mem Stall cycles per instruction) x Clock cycle time
- CPUtime = IC x (CPIexecution + Mem accesses per instruction x Miss rate x Miss penalty) x Clock cycle time
- Misses per instruction = Memory accesses per instruction x Miss rate
- CPUtime = IC x (CPIexecution + Misses per instruction x Miss penalty) x Clock cycle time

MK MORGAN KAUFMANN

Cache Performance

- Assuming the following execution and cache parameters:
  - Cache miss penalty = 50 cycles
  - Normal instruction execution CPI ignoring memory stalls = 2.0 cycles
  - Miss rate = 2%
  - Average memory references/instruction = 1.33
- CPU time = IC x [CPI execution + Memory accesses/instruction x Miss rate x Miss penalty] x Clock cycle time
- CPUtime with cache = IC x (2.0 + (1.33 x 2% x 50)) x clock cycle time
- = IC x 3.33 x Clock cycle time
- *Lower CPI execution increases the impact of cache miss clock cycles*

MK MORGAN KAUFMANN

Cache Performance

- Suppose a CPU executes at Clock Rate = 200 MHz (5 ns per cycle) with a single level of cache.
- CPIexecution = 1.1
- Instruction mix: 50% arith/logic, 30% load/store, 20% control
- Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles.
- CPI = $\text{CPI}_{\text{execution}}$ + mem stalls per instruction
- Mem Stalls per instruction =
- Mem accesses per instruction x Miss rate x Miss penalty
- Mem accesses per instruction = 1 + .3 = 1.3
- Mem Stalls per instruction = 1.3 x .015 x 50 = 0.975
- CPI = 1.1 + .975 = 2.075
- The ideal memory CPU with no misses is 2.075/1.1 = 1.88 times faster

Cache Performance

- Suppose for the previous example we double the clock rate to 400 MHZ, how much faster is this machine, assuming similar miss rate, instruction mix?
- Since memory speed is not changed, the miss penalty takes more CPU cycles:
- Miss penalty = 50 x 2 = 100 cycles.
- CPI = 1.1 + 1.3 x .015 x **100** = 1.1 + 1.95 = 3.05
- Speedup = (CPIold x Cold)/ (CPInew x Cnew)
- = 2.075 x 2 / 3.05 = 1.36
- The new machine is only 1.36 times faster rather than 2

times faster due to the increased effect of cache misses.

- *CPUs with higher clock rate, have more cycles per cache miss and more memory impact on CPI.*

Cache Performance

- Suppose a CPU uses separate level one (L1) caches for instructions and data (Harvard memory architecture) with different miss rates for instruction and data access:
  - $CPI_{execution}$ = 1.1
  - Instruction mix: 50% arith/logic, 30% load/store, 20% control
  - Assume a cache miss rate of 0.5% for instruction fetch and a cache data miss rate of 6%.
  - A cache hit incurs no stall cycles while a cache miss incurs 200 stall cycles for both memory reads and writes. Find the resulting CPI using this cache? How much faster is the CPU with ideal memory?

CPI = $CPI_{execution}$ + mem stalls per instruction

Mem Stall cycles per instruction = Instruction Fetch Miss rate x Miss Penalty + Data Memory Accesses Per Instruction x Data Miss Rate x Miss Penalty

Mem Stall cycles per instruction = 1 x 0.5/100 x 200 + 0.3 x 6/100 x 200 = 1 + 3.6 = 4.6

CPI = $CPI_{execution}$ + mem stalls per instruction = 1.1 + 4.6 = 5.7

The CPU with ideal cache (no misses) is 5.7/1.1 = 5.18 times faster

With no cache the CPI would have been = 1.1 + 1.3 X 200 = 261.1

Cache Performance

Size	Instruction cache	Data cache	Unified cache
1 KB	3.06%	24.61%	13.34%
2 KB	2.26%	20.57%	9.78%
4 KB	1.78%	15.94%	7.24%
8 KB	1.10%	10.19%	4.57%
16 KB	0.64%	6.47%	2.87%
32 KB	0.39%	4.82%	1.99%
64 KB	0.15%	3.77%	1.35%
128 KB	0.02%	2.88%	0.95%

Write Policy

1 Write Though: Data is written to both the cache block and to a block of main memory.

- The lower level always has the most updated data; an important feature for I/O and multiprocessing.
- Easier to implement than write back.
- A write buffer is often used to reduce CPU write stall while data is written to memory.

2 Write back: Data is written or updated only to the cache block. The modified or dirty cache block is written to main memory when it's being replaced from cache.

- Writes occur at the speed of cache
- A status bit called a dirty or modified bit, is used to indicate whether the block was modified while in cache; if not the block is not written back to main memory when replaced.
- Uses less memory bandwidth than write through.

Write Policy

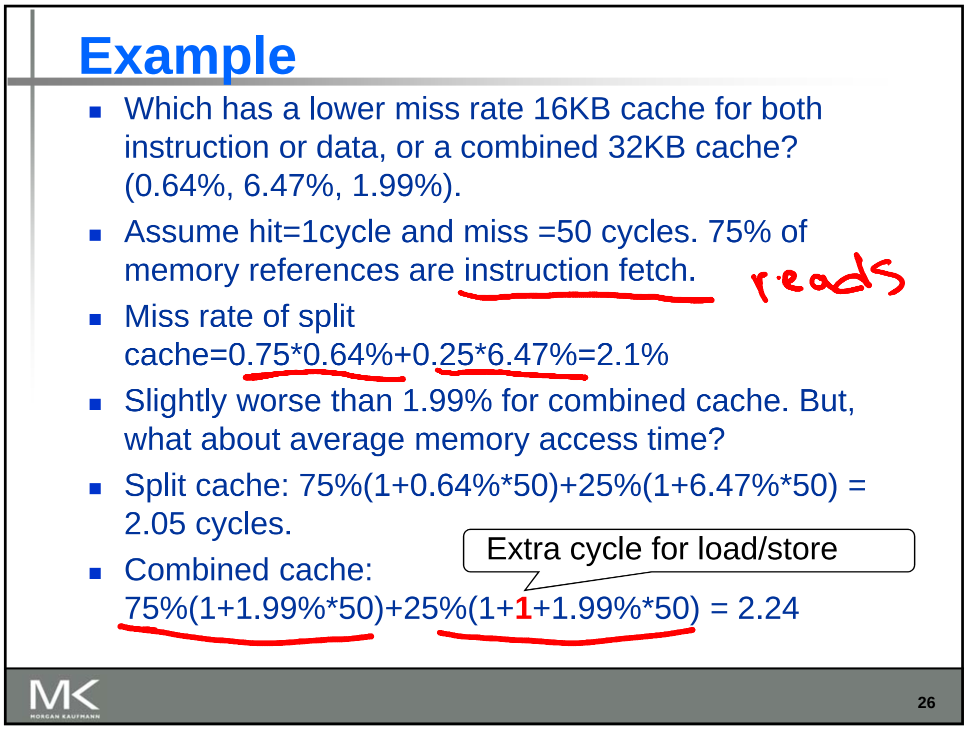
Write Allocate:

The cache block is loaded on a write miss followed by write hit actions.

No-Write Allocate:

The block is modified in the lower level (lower cache level, or main

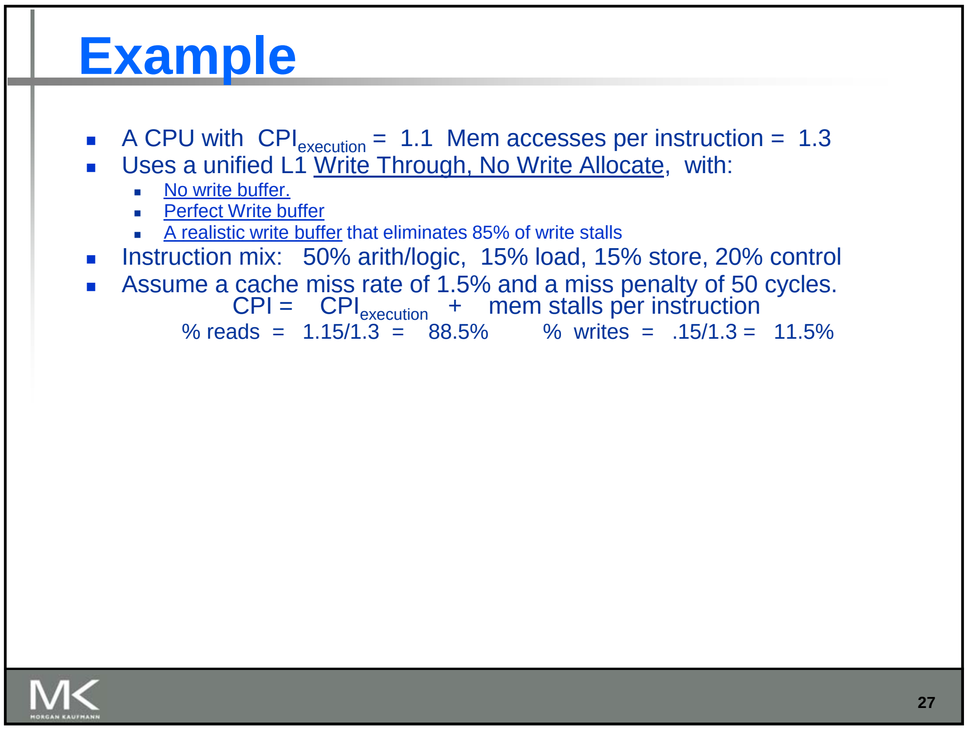
memory) and not loaded into cache.



Example

- Which has a lower miss rate 16KB cache for both instruction or data, or a combined 32KB cache? (0.64%, 6.47%, 1.99%).
- Assume hit=1cycle and miss =50 cycles. 75% of memory references are instruction fetch. reads
- Miss rate of split cache=0.75*0.64%+0.25*6.47%=2.1%
- Slightly worse than 1.99% for combined cache. But, what about average memory access time?
- Split cache: 75%(1+0.64%*50)+25%(1+6.47%*50) = 2.05 cycles.
- Combined cache: 75%(1+1.99%*50)+25%(1+1+1.99%*50) = 2.24

Extra cycle for load/store



Example

- A CPU with $CPI_{execution}$ = 1.1 Mem accesses per instruction = 1.3
- Uses a unified L1 Write Through, No Write Allocate, with:
  - No write buffer.
  - Perfect Write buffer
  - A realistic write buffer that eliminates 85% of write stalls
- Instruction mix: 50% arith/logic, 15% load, 15% store, 20% control
- Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles.

  CPI = $CPI_{execution}$ + mem stalls per instruction

  % reads = 1.15/1.3 = 88.5%   % writes = .15/1.3 = 11.5%

Example

- A CPU with $CPI_{execution}$ = 1.1 uses a unified L1 with with write back, with write allocate, and the probability a cache block is dirty = 10%
- Instruction mix: 50% arith/logic, 15% load, 15% store, 20% control
- Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles.

1.3 mem refs / inst

50+50

$$\frac{1.5}{100}\left(1.3\times 50\times 0.9+1.3\times 0.1\times 100\right)$$

Example

- CPU with $CPI_{execution}$ = 1.1 running at clock rate = 500 MHz
- 1.3 memory accesses per instruction.
- L_1 cache operates at 500 MHz with a miss rate of 5%
- L_2 cache operates at 250 MHz with local miss rate 40%, (T_2 = 2 cycles)
- Memory access penalty, M = 100 cycles. Find CPI.

$$1.3\left(\frac{5}{100}\times 0.6\times 2+\frac{5}{100}\times 0.4\times 100\right)$$

Example

- CPU with $CPI_{execution}$ = 1.1 running at clock rate = 500 MHz
- 1.3 memory accesses per instruction.
- For L_1 :
  - Cache operates at 500 MHz with a miss rate of 1-H1 = 5%
  - Write though to L_2 with perfect write buffer with write allocate
- For L_2:
  - Cache operates at 250 MHz with local miss rate 1- H2 = 40%, (T_2 = 2 cycles)
  - Write back to main memory with write allocate
  - Probability a cache block is dirty = 10%
- Memory access penalty, M = 100 cycles. Find CPI.

$$0.05\left(0.6 \times 2 + 0.4 \times 0.9 \times 100 + 0.4 \times 0.1 \times 200\right)$$

Example

- CPU with $CPI_{execution}$ = 1.1 running at clock rate = 500 MHz
- 1.3 memory accesses per instruction.
- L_1 cache operates at 500 MHz with a miss rate of 5%
- L_2 cache operates at 250 MHz with a local miss rate 40%, (T_2 = 2 cycles)
- L_3 cache operates at 100 MHz with a local miss rate 50%, (T_3 = 5 cycles)
- Memory access penalty, M= 100 cycles. Find CPI.

HW

Cache Miss

- Compulsory: The very first access to a block is always a miss– Ocurs even if you have an infinite cache
- Capacity: The cache is not big enough to hold all the blocks required for the execution of the program– A bigger cache helps
- Conflict: If not a fully associative, a block may be discarded and brought back again.

Memory Hierarchy Basics

- Six basic cache optimizations:
  - Larger block size
    - Reduces compulsory misses
    - Increases capacity and conflict misses, increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Increases hit time, increases power consumption
  - Higher associativity
    - Reduces conflict misses
    - Increases hit time, increases power consumption
  - Higher number of cache levels
    - Reduces overall memory access time
  - Giving priority to read misses over writes
    - Reduces miss penalty
  - Avoiding address translation in cache indexing
    - Reduces hit time

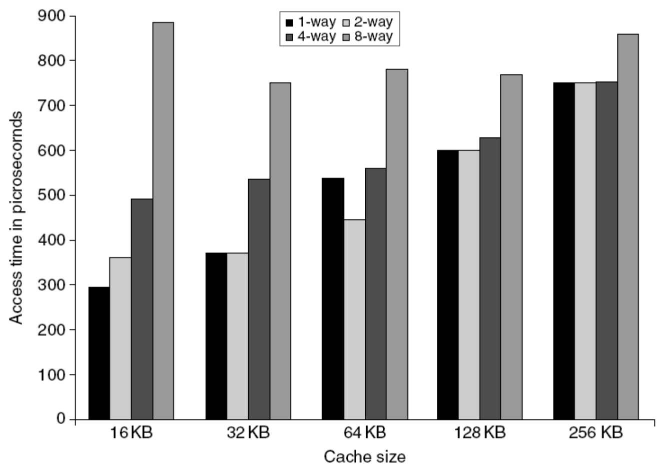
Ten Advanced Optimizations

- Small and simple first level caches
- Way Prediction
- Pipelined caches
- Non-blocking cache
- Multibanked cache
- Critical word first
- Merging write buffer
- Compiler optimization
- Hardware prefetching
- Compiler prefetching

Small and Simple

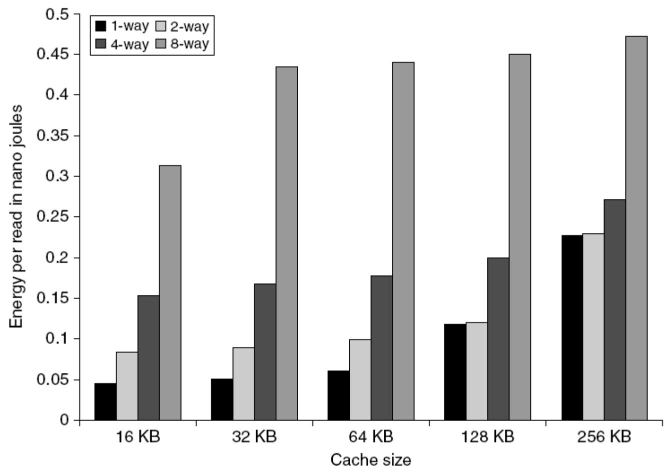
- No mux in the critical path of a direct mapped cache.
- Bigger cache means more energy.
- CACTI – An idea for the project/paper review
- Many processors takes at least 2 clock cycles to access the cache, longer hit time may not be that critical
- The use of a virtual index cache, limits the cache size to page size $\times$ associativity (recently a trend to increase associativity).

L1 Size and Associativity



Access time vs. size and associativity

L1 Size and Associativity



Energy per read vs. size and associativity

Way Prediction

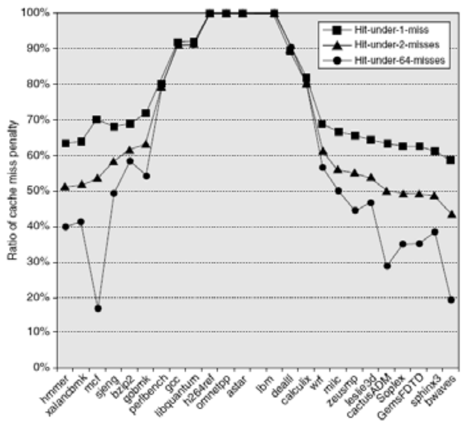
- To improve hit time, predict the way to pre-set mux
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- Extend to predict block as well
  - “Way selection”
  - Increases mis-prediction penalty

Pipelining Cache

- Pipeline cache access to improve bandwidth
  - Examples:
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles
- Increases branch miss-prediction penalty (longer pipeline).
- Makes it easier to increase associativity

Nonblocking Caches

- For out-of-order execution (later on this point).
- Allow hits before previous misses complete
  - "Hit under miss"
  - "Hit under multiple miss"
- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty



Single core i7 using SPEC2006

Multibanked Caches

- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2
- Interleave banks according to block address

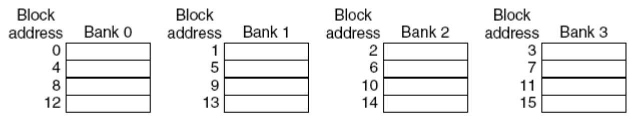


Figure 2.6 **Four-way interleaved cache banks using block addressing.** Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

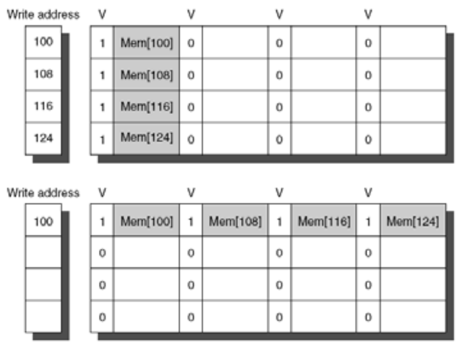
Critical Word First, Early Restart

- Critical word first
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives
- Early restart
  - Request words in normal order
  - Send missed work to the processor as soon as it arrives

- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched

Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer
- Do not apply to I/O addresses



No write buffering

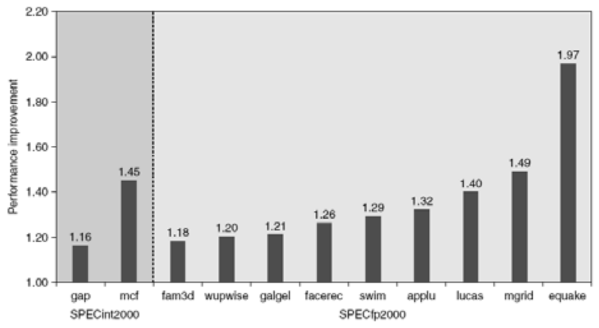
Write buffering

Compiler Optimizations

- Loop Interchange
  - Swap nested loops to access memory in sequential order (row major access)
- Blocking
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Requires more memory accesses but improves locality of accesses

Hardware Prefetching

- Fetch two blocks on miss (include next sequential block) (the 2nd one goes to instruction stream buffer, must be checked if found do not go to cache).



Pentium 4 Pre-fetching

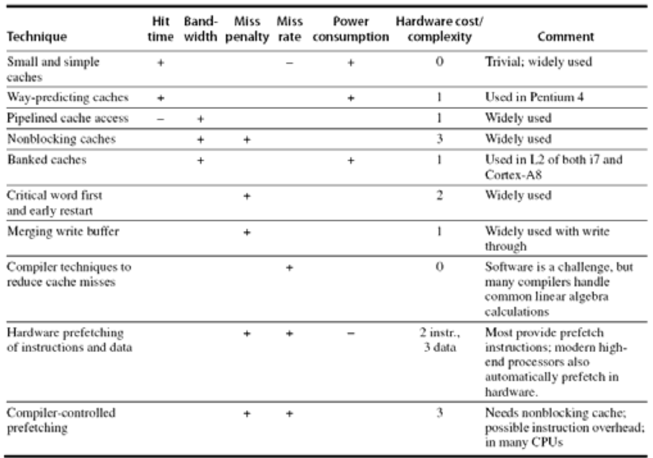
Compiler Prefetching

- Insert prefetch instructions before data is needed
- Non-faulting: prefetch doesn't cause exceptions

- Register prefetch
  - Loads data into register
- Cache prefetch
  - Loads data into cache

- Combine with loop unrolling and software pipelining

Summary



Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined cache access	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	1	Used in L2 of both i7 and Cortex-A8
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware.
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs

Figure 2.11 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor, – means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.