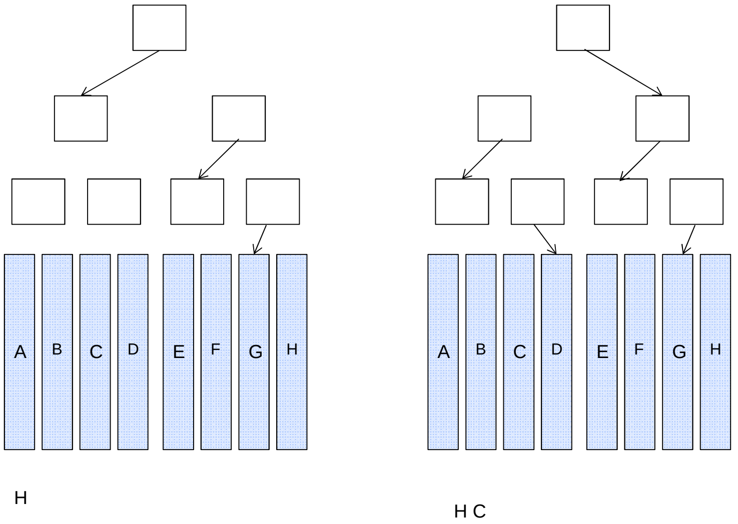
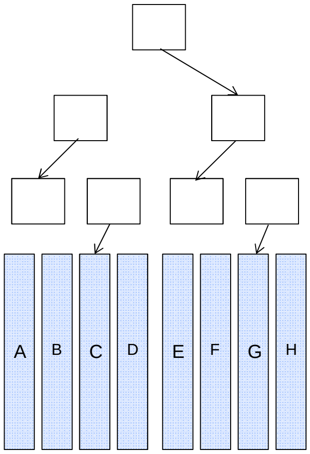
LRU

- A list to keep track of the order of access to every block in the set.
- The least recently used block is replaced (if needed).
- How many bits we need for that?

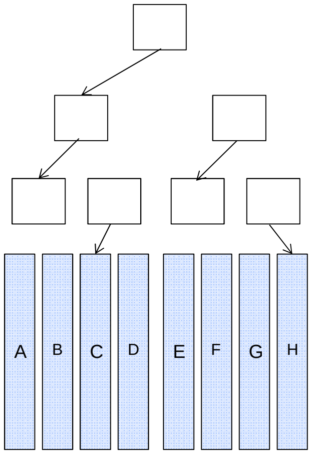
Pseudo LRU



Psuedo LRU

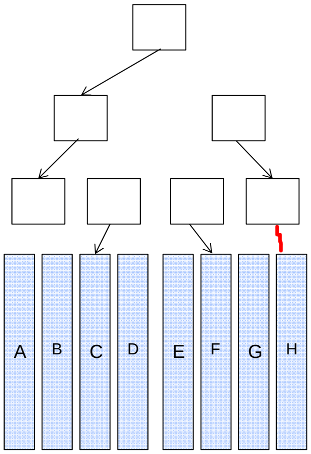


H C D

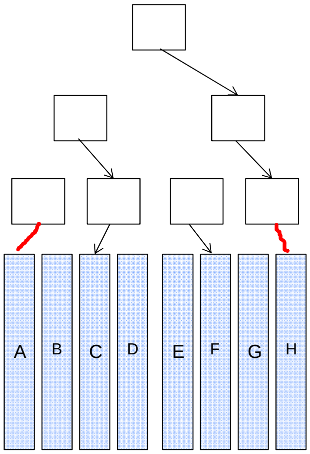


H C D G

Psuedo LRU

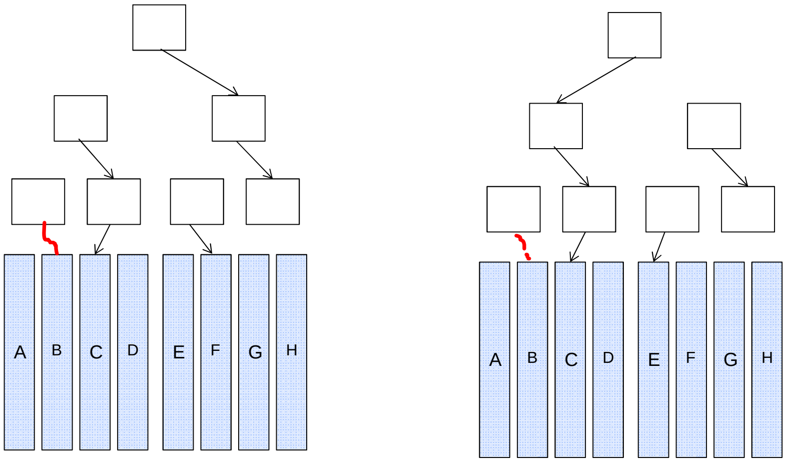


H C D G E



H C D G E B

Psuedo LRU



H C D G E B A

H C D G E B F

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 <u>Write Through, No Write Allocate</u>, with:
  - <u>No write buffer.</u>
  - <u>Perfect Write buffer</u>
  - <u>A realistic write buffer</u> 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 <u>write back</u>, with <u>write allocate</u>, and the <u>probability a cache block is dirty = 10%</u>
- 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

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

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.

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

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

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

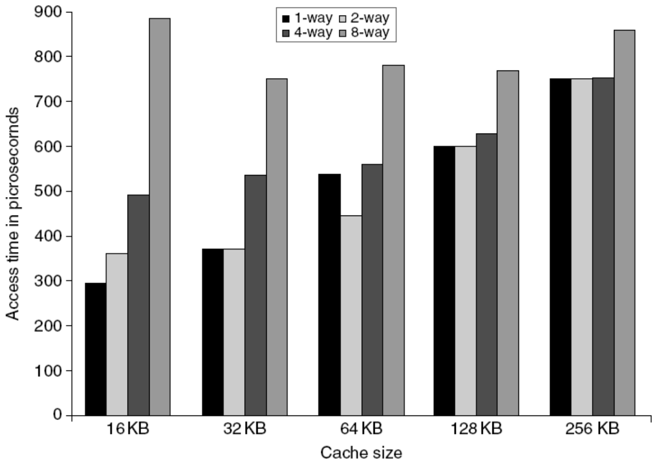
Advanced Optimizations

- 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

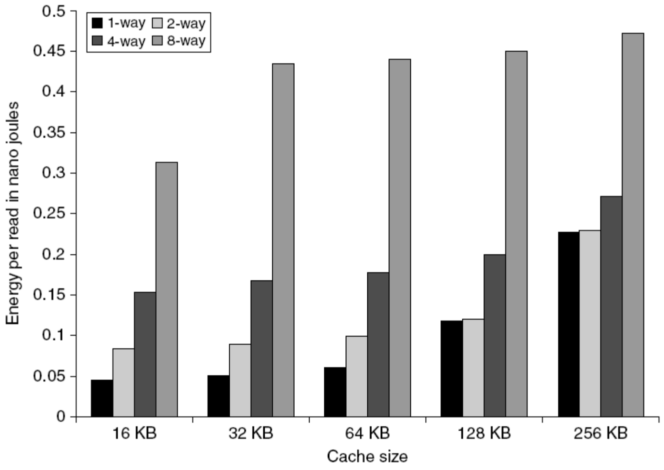
- No mux in the critical path of a direct mapped cache.
- Bigger cache means more energy.
- CACTI – An idea for the project/paper review
- Many processors takes at least 2 clock cycles to access the cache, longer hit time may not be that critical
- The use of a virtual index cache, limits the cache size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

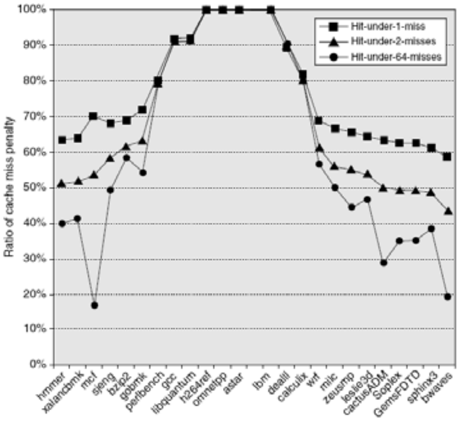
- To improve hit time, predict the way to pre-set mux
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- Extend to predict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

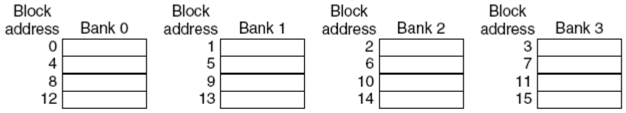
- For out-of-order execution (later on this point).
- Allow hits before previous misses complete
  - “Hit under miss”
  - “Hit under multiple miss”
- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty



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



Block address	Bank 0	Block address	Bank 1	Block address	Bank 2	Block address	Bank 3
0		1		2		3	
4		5		6		7	
8		9		10		11	
12		13		14		15	

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.

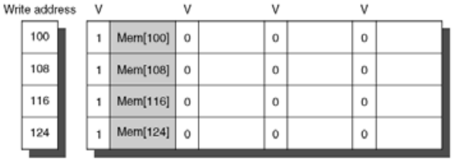
Critical Word First, Early Restart

- 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

---

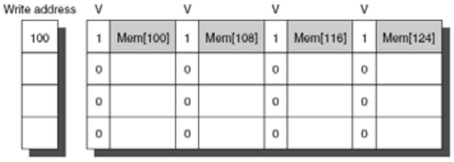
Merging Write Buffer

- 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



Write address	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

No write buffering



Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

Write buffering

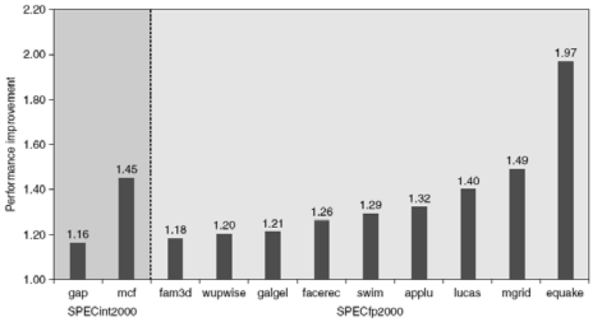
Compiler Optimizations

- 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

Advanced Optimizations

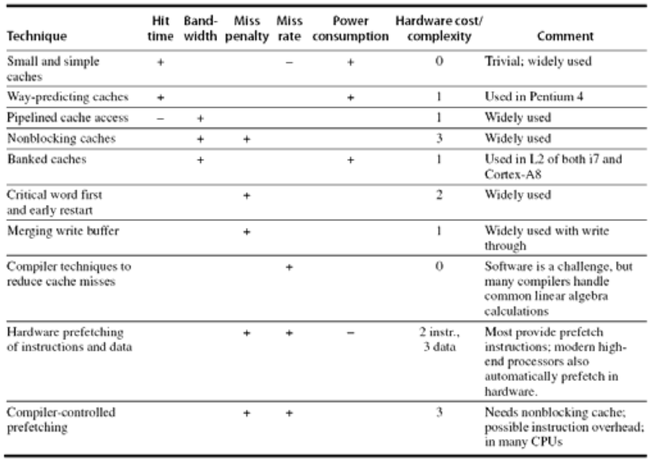
- Insert prefetch instructions before data is needed
- Non-faulting: prefetch doesn't cause exceptions

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

- Combine with loop unrolling and software pipelining

Summary

Advanced Optimizations



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.