
Lecture 12: Memory Hierarchy Design Contd.

Kunle Olukotun
Gates 302
kunle@ogun.stanford.edu

<http://www-leland.stanford.edu/class/ee282h/>

1

Cache Problem

- Given the following data
 - » 1 instruction reference per instruction
 - » 0.27 loads/instruction
 - » 0.13 stores/instruction
 - » Memory access time = 4 cycles + # words/block
 - » A 64KB cache with 4 W block size has a miss rate of 1.7%
 - » Base CPI of 1.5
- Suppose the cache uses a write through, no write allocate, write around write strategy without a write buffer. How much faster would the machine be with a perfect write buffer?

2

No Write Buffer

3

Perfect Write Buffer

4

Improving Cache Performance

- $AMAT = \text{hit time} + \text{miss rate} \times \text{miss penalty}$
 - » Reduce any of these to improve performance
- Start with miss rate
- Miss rate data needs to be interpreted carefully
 - » **Very** dependent on application, language, operating system
 - multitasking has a big effect
 - » Miss rates for a given cache design tend to increase over time!

5

Understanding where misses come from: An intuitive model

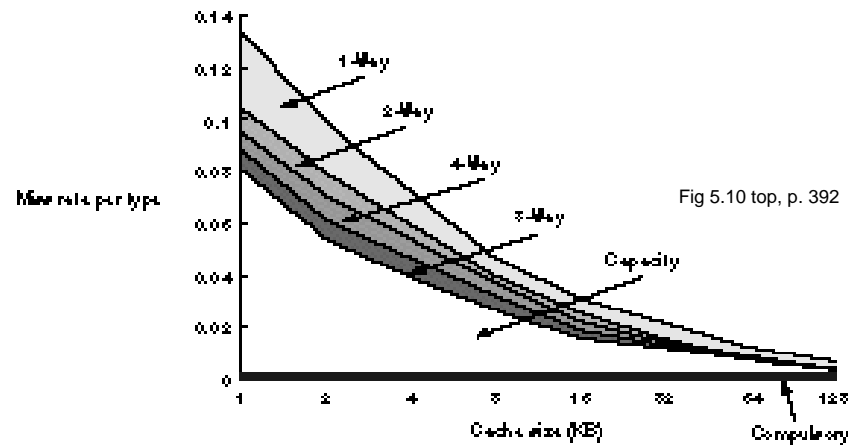
- Compulsory: unavoidable first reference to a block
- Capacity: misses caused because the cache is too small
- Conflict: misses caused by mapping conflicts in the cache

Type	Measure by	Reduce by	Impact
Compulsory (C1)	$C1 = \text{MR of infinite cache}$	increase block size	C2 and C3 can increase
Capacity (C2)	$C2 = \text{MR of fully associative cache} - C1$	increase cache size	increased cycle time and cost
Conflict (C3)	$C3 = \text{MR of set associative cache} - C2 - C3$	increase associativity	increased cycle time and cost

- Just a way to think about misses; reality is more complicated
 - » Increasing cache size can eliminate some conflict misses.

6

Total Miss Rate



- Cache rules of thumb
 - » cache size $N = 2\text{-way size } N/2$
 - » double cache size removes 30% of misses

7

Miss rate vs. Cache size

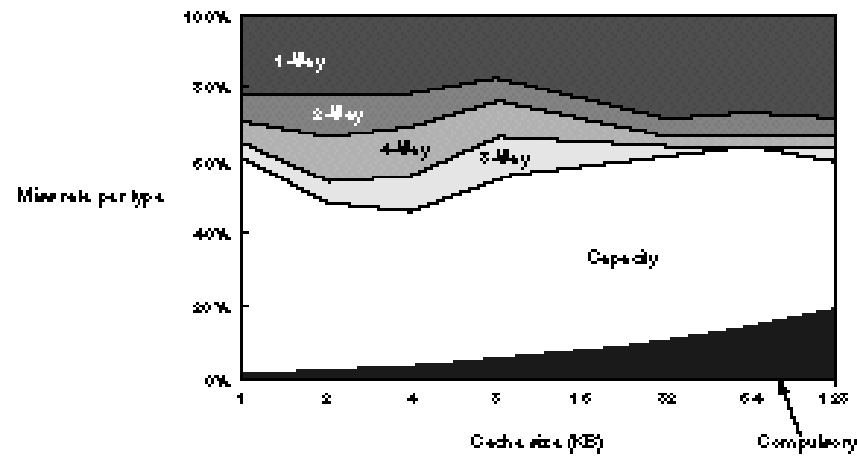


Fig 5.10 bottom, p. 392

8

Working Set Behavior

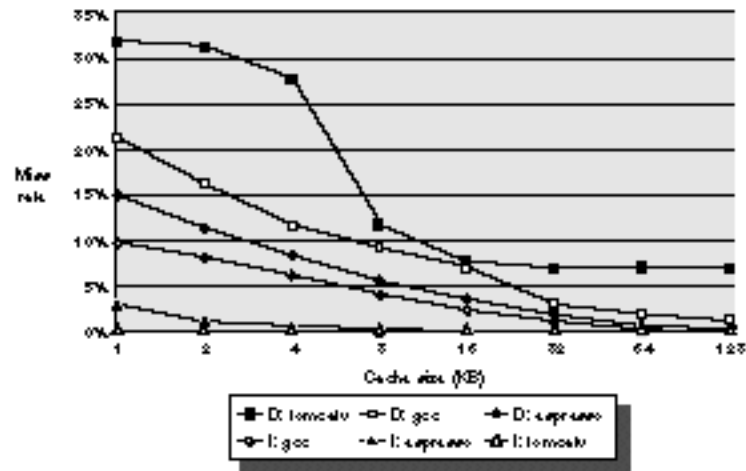
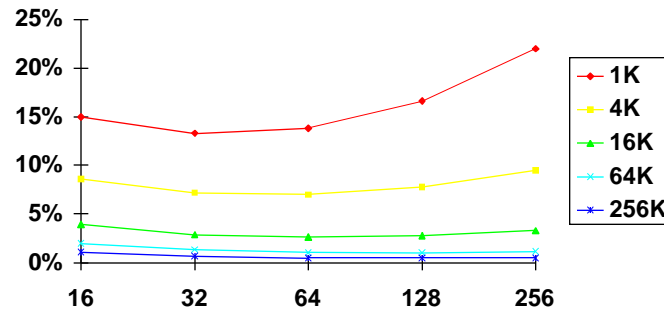


Fig 5.50 , p. 468

9

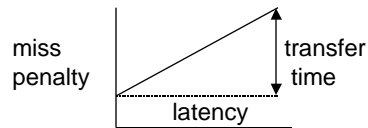
Reducing misses: Larger block size vs. Miss Penalty

- Increases miss penalty
- Decreases miss rate because data is pre-fetched
 - » Effect reverses when a lot of prefetched data is not used (conflict misses)
- Reduces overhead (tag as % of data)
- Better for quick load of compulsory misses after task switch
- Large block sizes are better for I-caches than D-caches

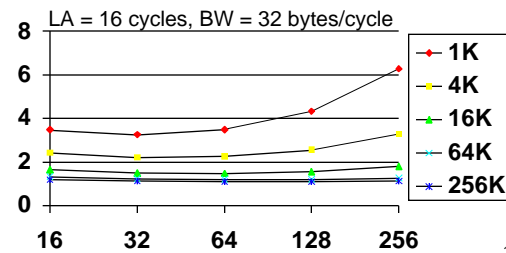
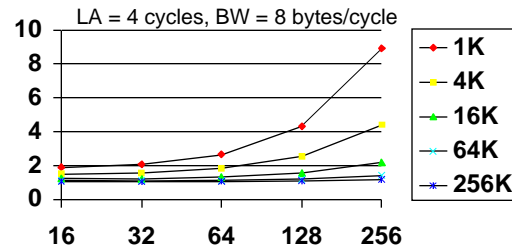


10

Reducing misses: Larger block size vs. AMAT



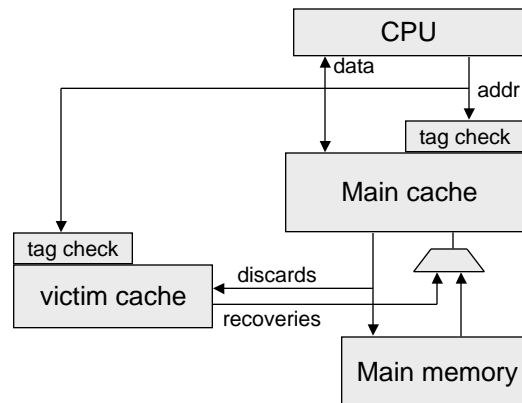
- LA = latency (for 1st word)
- BW = transfer rate per byte
- AMAT = $1 + \text{missrate} \times (\text{LA} + \text{blksize}/\text{BW})$
- High LA and high BW favor large block sizes



11

Reducing misses: Victim cache

- Small, fully associative cache containing recently discarded blocks
- Lookup in parallel with main cache
- Think of it as higher associativity for a few of the cache lines
- Works best with small direct-mapped caches.
- Can remove 50% of conflict misses in 4KB DM with only 4 entries!



12

Reducing misses: Pseudo-associative caches

- Two-step cache check, to achieve:
 - » the hit speed (almost) of a direct-mapped cache
 - » the miss rate (almost) of a 2-way associative cache
- After a miss as a direct-mapped cache at location $b(addr)$, make another check at a related location $f(addr)$.
- Swap to make the most recent hit the first to be checked.



Hit/miss rates

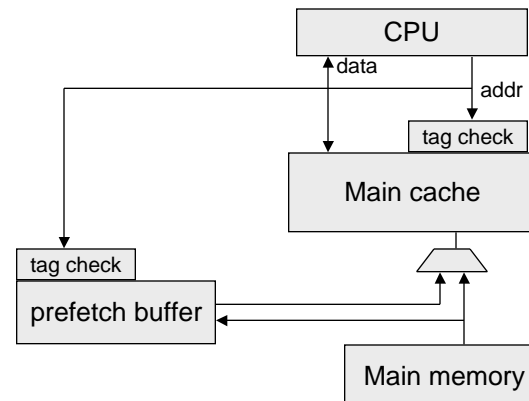
	← 100 % →		
direct mapped	hit	miss	
2-way	hit	miss	
pseudo	hit 1	hit 2	miss

$$AMAT = 1 + (\text{direct-mapped miss rate} \times 2\text{nd check cost}) + (2\text{-way miss rate} \times \text{miss penalty})$$

13

Reducing misses: Hardware prefetching

- On a miss, fetch the requested block, and the next block(s).
- Store the next block "near" the cache, and prefetch again when used.
- Think of it as a bigger blocksize for one or a few of the cache lines.
- Can be used for instructions or data.
- Can remove 43% of the misses with 4 prefetch blocks in 4 KB DM!



14

Reducing misses: Software techniques

- Software prefetching
 - » Load-to-register or load-to-cache instructions
 - » Should be non-blocking to allow other cache references to proceed (more later)

```

for (i = 0; i < 1000; i++) {
    sum = sum + A[i];
}

for (i = 0; i < 1000; i++) {
    prefetch A[i+10];
    sum = sum + A[i];
}

```

- Change the memory reference pattern
 - » For code:
 - make the inline (non-branch) path the common case
 - isolate all exception-handling code together
 - » For data: Improve locality
 - Merge arrays which are accessed in parallel
int a[100], b[100]; becomes **struct {int a; int b;} ab[100];**
 - Interchange loops for 2-dimensional arrays if access pattern is not sequential (row-wise vs. column-wise)
 - Fuse two consecutive loops through an array into one.
 - For large arrays, operate on sub-arrays rather than row-wise or column-wise (blocking)

15

Performance of Software Techniques

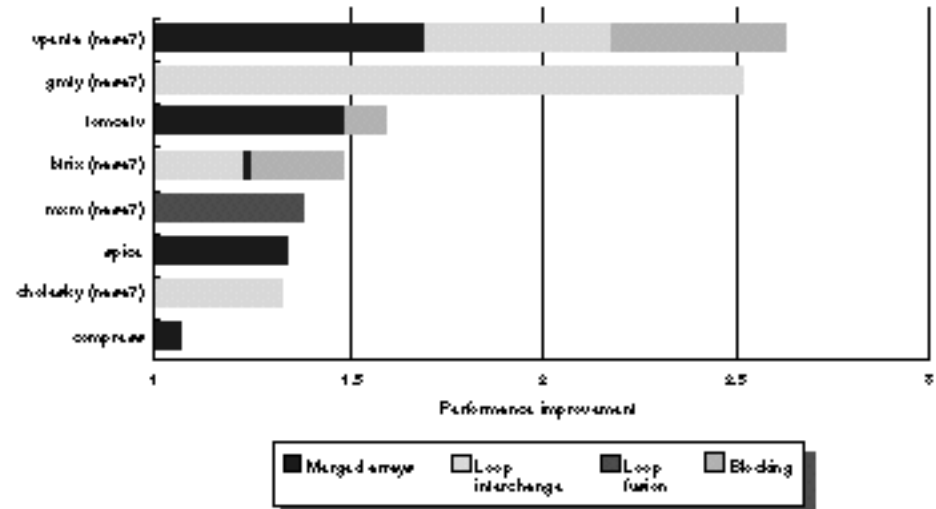


Fig 5.17, p. 392

16

Reducing the miss penalty: Hardware tricks

- Give priority to reads over writes
 - » Be careful to check for RAW hazards in the store buffer to memory
- Subblock allocation: only read part of a block
 - » saves on tags and comparators compared to more associativity
 - » works well with write-allocate no-fetch write miss policy
- Get the CPU what it needs as quickly as possible:
 - » early restart: as soon as the needed word arrives, send it and continue fetching
 - » critical word first (wrap-around fetch): ask for the needed word first, and continue fetching
- Nonblocking Cache
 - » Don't stall CPU for miss!

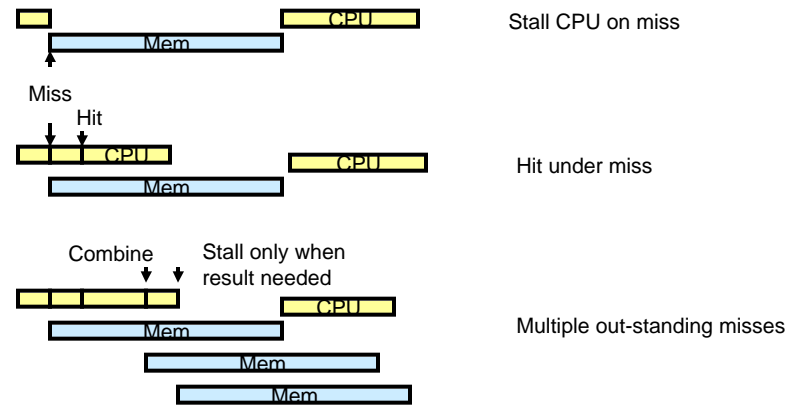
17

Nonblocking Cache

- First reference
 - » Lockup free cache
 - » Kroft, ISCA, 1981
- Hardware mechanisms that provide outstanding misses
 - » One outstanding miss
 - “hit under miss”
 - hide miss latency with CPU execution and data references that hit
 - Used in PA7100, DEC Alpha 21064
 - » Multiple outstanding misses
 - overlap multiple misses
 - requires more hardware than hit under miss
 - requires the next level up in the memory hierarchy to be pipelined
 - Used in UltraSPARC, R10000, DEC Alpha 21164
- Performance benefits from non-blocking loads depend on instruction scheduling or out-of-order execution and amount of ILP in application

18

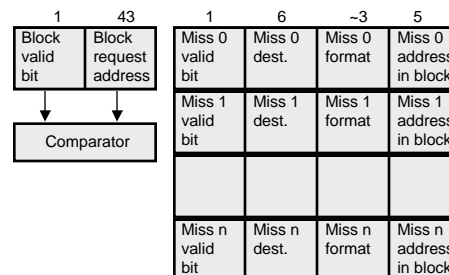
Nonblocking Cache Potential



19

Nonblocking Cache Hardware

- NBC terminology
 - » *Primary miss* is the first to a main memory block
 - » *Secondary misses* are the subsequent misses to a block
 - » *Structural-stall misses* are the subsequent misses to a block that stall because of resource contention
- Miss Status Holding Registers (MSHRs)
 - » Holds enough information on all outstanding misses to complete the load
 - » Primary and secondary misses can use the same MSHR



20

Nonblocking Cache Performance

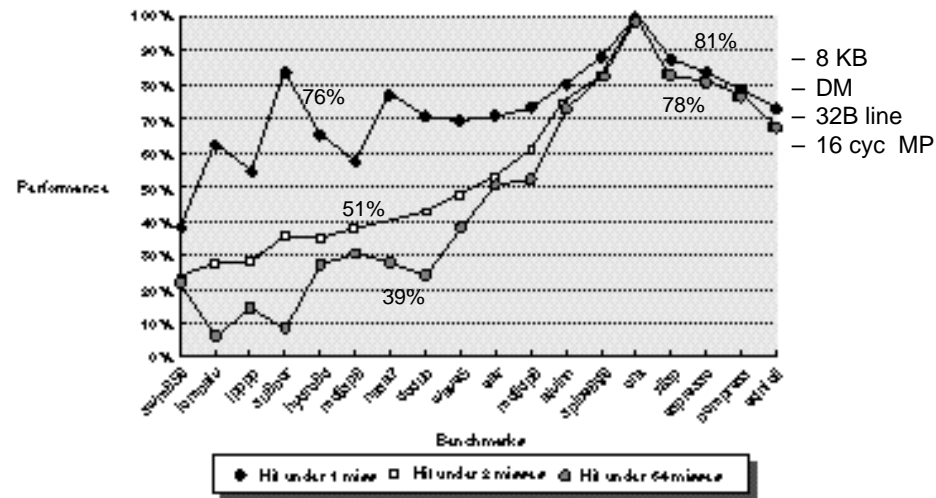


Fig 5.22, p. 415 21

Reducing the miss penalty: Two-level caches

- $AMAT = L1 \text{ hit time} + L1 \text{ miss rate} \times (L2 \text{ hit time} + L2 \text{ miss rate} \times L2 \text{ miss penalty})$
- This is based on *local miss rate*: misses/accesses to this cache
- The *global miss rate*: misses/ memory accesses by CPU
- The local miss rate of L2 caches is typically quite bad, since L1 has caught most locality.
 - » Make it big, with big block size and high associativity to lower the miss rate, since the hit time doesn't matter so much
- Want *multi-level inclusion*: everything at level i is also in level $i+1$
 - » make analysis easier: solo miss rate of L2 = overall (global) miss rate = (local) miss rate of L1 \times local miss rate of L2
 - » I/O and multiprocessors only need to look at higher levels
 - » Will be true if (not only if):
 - same blocksize and associativity
 - L2 is larger than L1

L2 Miss Rates

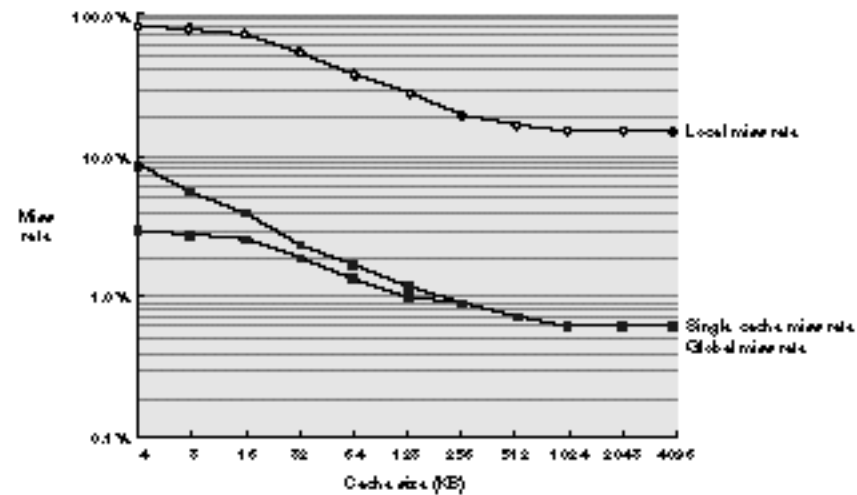


Fig 5.23, p. 418

23

Reducing the hit time: Hardware tricks

- Use direct-mapped cache
 - » no multiplexer in the path
 - » data can be speculatively used while tag check is in progress
- Write hits take longer because tag must be checked before writing (read and write are in sequence). To fix, pipeline the writes:
 - » WRITE1: readtag writecache
 - » WRITE2: readtag writecache
 - » (Ignore read hits, but watch for RAW hazards!)
 - » Used by 21064, VAX 8800
- For write-through write-invalidate direct-mapped caches only: always write regardless of tag!
 - » use for subblock caches (multiple valid bits)
 - » if the tag was right, it was the right block after all
 - » if the tag was wrong, change the tag and invalidate other words (memory still has a good copy)
 - » Do the write-through

24

Cache Improvement Summary

- Figure 5.29 from textbook