

CS184b: Computer Architecture (Abstractions and Optimizations)

Day 12: April 27, 2005
Caching Introduction



Caltech CS184 Spring2005 -- DeHon

Today

- Memory System
 - Issue
 - Structure
 - Idea
 - Cache Basics

Caltech CS184 Spring2005 -- DeHon

2

Memory and Processors

- Memory used to compactly store
 - state of computation
 - description of computation (instructions)
- Memory access latency impacts performance
 - timing on load, store
 - timing on instruction fetch

Caltech CS184 Spring2005 -- DeHon

3

Issues

- Need big memories:
 - hold large programs (many instructions)
 - hold large amounts of state
- Big memories are slow
- Memory takes up areas
 - want dense memories
 - densest memories not fast
 - fast memories not dense
- Memory capacity needed not fit on die
 - inter-die communication is slow

Caltech CS184 Spring2005 -- DeHon

4

Problem

- Desire to contain problem
 - implies large memory
- Large memory
 - implies slow memory access
- Programs need frequent memory access
 - e.g. 20% load operations
 - fetch required for every instruction
- **Memory is the performance bottleneck?**
 - Programs run slow?

Caltech CS184 Spring2005 -- DeHon

5

Opportunity

- Architecture mantra:
 - exploit structure in typical problems
- What structure exists?

Caltech CS184 Spring2005 -- DeHon

6

Memory Locality

- What percentage of accesses to unique addresses
 - addresses distinct from the last N unique addresses

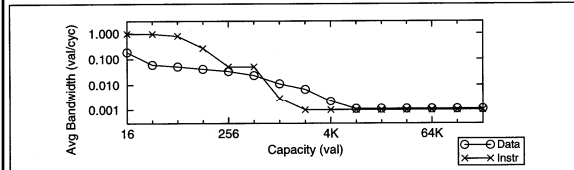


Figure 3-1 Bandwidth spectrums for jpeg.

Caltech CS184 Spring2005 -- DeHon [Huang+Shen, Intrinsic BW, ASPLOS 7]

[from CS184a]

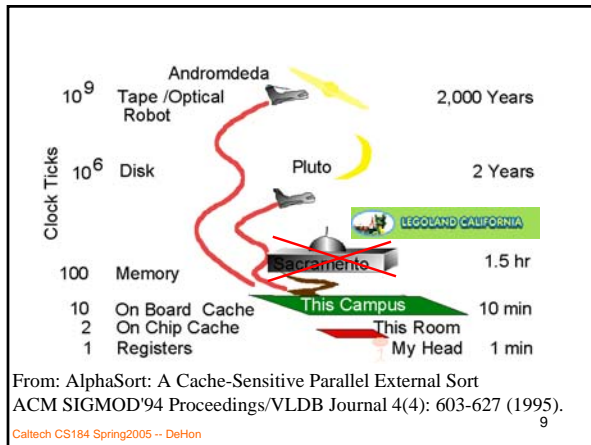
Hierarchy/Structure Summary

- “Memory Hierarchy” arises from area/bandwidth tradeoffs
 - Smaller/cheaper to store words/blocks
 - (saves routing and control)
 - Smaller/cheaper to handle long retiming in larger arrays (reduce interconnect)
 - High bandwidth out of registers/shallow memories

λ^2	DRAM	SRAM	RF bit	FF/RF	RF \times 1	XC	In FF	net FF	FF/LUT
100	100	1200	2K	5K	40K	40K	75K	200K	800K
bw/cap.	$1/10^7$	$1/10^5-10^3$		$1/100$	$1/100$	$1/16$	$1/4$	$1/1$	$1/1$

Caltech CS184 Spring2005 -- DeHon

8



From: AlphaSort: A Cache-Sensitive Parallel External Sort
ACM SIGMOD'94 Proceedings/VLDB Journal 4(4): 603-627 (1995).

Caltech CS184 Spring2005 -- DeHon

9

Opportunity

- Small memories are fast
- Access to memory is not random
 - temporal locality
 - short and long retiming distances
- Put commonly/frequently used data (instructions) in small memory

Caltech CS184 Spring2005 -- DeHon

10

Memory System Idea

- Don't build single, flat memory
- Build a hierarchy of speeds/sizes/densities
 - commonly accessed data in fast/small memory
 - infrequently used data in large/dense/cheap memory
- Goal
 - achieve speed of small memory
 - with density of large memory

Caltech CS184 Spring2005 -- DeHon

11

Hierarchy Management

- Two approaches:
 - explicit data movement
 - register file
 - overlays
 - transparent/automatic movement
 - invisible to model

Caltech CS184 Spring2005 -- DeHon

12

Opportunity: Model

- Model is simple:
 - read data and operate upon
 - timing not visible
- Can vary timing
 - common case fast (in small memory)
 - all cases correct
 - can answered from larger/slower memory

Caltech CS184 Spring2005 -- DeHon

13

Cache Basics

- Small memory (cache) holds commonly used data
- Read goes to cache first
- If cache holds data
 - return value
- Else
 - get value from bulk (slow) memory
- Stall execution to hide latency
 - full pipeline, scoreboarding

Caltech CS184 Spring2005 -- DeHon

14

Cache Questions

- How manage contents?
 - decide what goes (is kept) in cache?
- How know what we have in cache?
- How make sure consistent ?
 - between cache and bulk memory

Caltech CS184 Spring2005 -- DeHon

15

Cache contents

- **Ideal:** cache should hold the N items that maximize the fraction of memory references which are satisfied in the cache
- **Problem:**
 - don't know future
 - don't know what values will be needed in the future
 - partially limitation of model
 - partially data dependent
 - halting problem
 - (can't say if will execute piece of code)

Caltech CS184 Spring2005 -- DeHon

16

Cache Contents

- Look for heuristics which keep most likely set of data in cache
- **Structure:** temporal locality
 - high probability that recent data will be accessed again
- **Heuristic goal:**
 - keep the last N references in cache

Caltech CS184 Spring2005 -- DeHon

17

Temporal Locality Heuristic

- Move data into cache on access (load, store)
- Remove "old" data from cache to make space

Caltech CS184 Spring2005 -- DeHon

18

“Ideal” Locality Cache

- Stores N most recent things
 - store any N things
 - know which N things accessed
 - know when last used

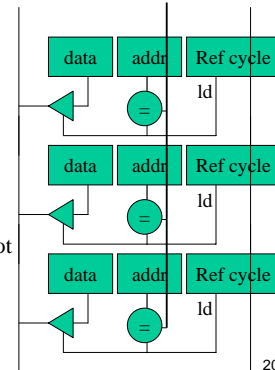


Caltech CS184 Spring2005 -- DeHon

19

“Ideal” Locality Cache

- Match address
- If matched,
 - update cycle
- Else
 - drop oldest
 - read from memory
 - store in newly free slot

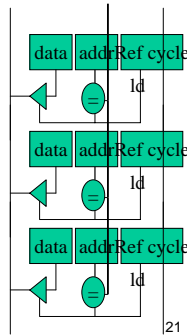


Caltech CS184 Spring2005 -- DeHon

20

Problems with “Ideal” Locality?

- Need $O(N)$ comparisons
- Must find oldest
 - (also $O(N)$?)
- Expensive



Caltech CS184 Spring2005 -- DeHon

21

Relaxing “Ideal”

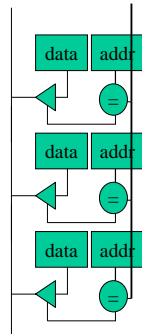
- Keeping usage (and comparing) expensive
- Relax:
 - Keep only a few bits on age
 - Don't bother
 - pick victim randomly
 - things have expected lifetime in cache
 - old things more likely than new things
 - if evict wrong thing, will replace
 - very simple/cheap to implement

Caltech CS184 Spring2005 -- DeHon

22

Fully Associative Memory

- Store both
 - address
 - data
- Can store any N addresses
- approaches ideal of “best” N things



Caltech CS184 Spring2005 -- DeHon

23

Relaxing “Ideal”

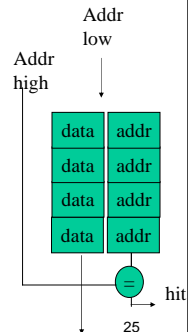
- Comparison for every address is expensive
- Reduce comparisons
 - deterministically map address to a small portion of memory
 - Only compare addresses against that portion

Caltech CS184 Spring2005 -- DeHon

24

Direct Mapped

- Extreme is a “direct mapped” cache
- Memory slot is $f(\text{addr})$
 - usually a few low bits of address
- Go directly to address
 - check if data want is there



Caltech CS184 Spring2005 -- DeHon

Direct Mapped Cache

- Benefit
 - simple
 - fast
- Cost
 - multiple addresses will need same slot
 - conflicts mean don't really have most recent N things
 - can have conflict between commonly used items

26

Caltech CS184 Spring2005 -- DeHon

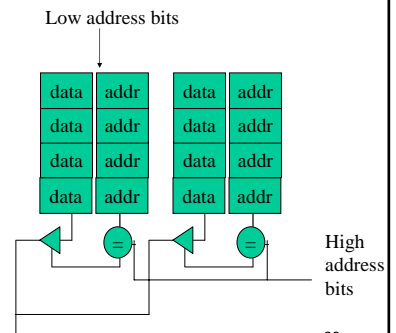
Set-Associative Cache

- Between extremes set-associative
- Think of M direct mapped caches
- One comparison for each cache
- Lookup in all M caches
- Compare and see if any have target data
- Can have M things which map to same address

27

Caltech CS184 Spring2005 -- DeHon

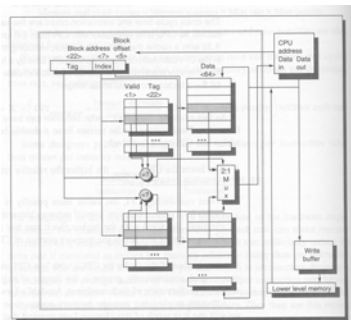
Two-Way Set Associative



28

Caltech CS184 Spring2005 -- DeHon

Two-way Set Associative



[Hennessy and Patterson 5.8e2] 29

Caltech CS184 Spring2005 -- DeHon

Set Associative

- More expensive than direct mapped
- Can decide expense
- Slower than direct mapped
 - have to mux in correct answer
- Can better approximate holding N most recently/frequently used things

30

Caltech CS184 Spring2005 -- DeHon

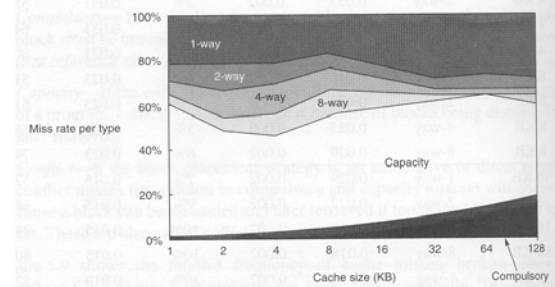
Classify Misses

- Compulsory
 - first reference
 - (any cache would have)
- Capacity
 - misses due to size
 - (fully associative would have)
- Conflict
 - miss because of limit places to put

Caltech CS184 Spring2005 -- DeHon

31

Set Associativity

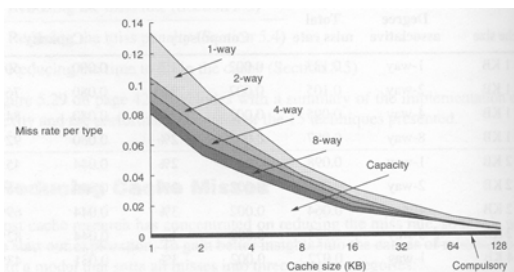


[Hennessy and Patterson 5.10e2]

Caltech CS184 Spring2005 -- DeHon

32

Absolute Miss Rates



[Hennessy and Patterson 5.10e2]

Caltech CS184 Spring2005 -- DeHon

33

Policy on Writes

- Keep memory consistent at all times?
 - Or cache+memory holds values?
- Write through:
 - all writes go to memory and cache
- Write back:
 - writes go to cache
 - update memory only on eviction

Caltech CS184 Spring2005 -- DeHon

34

Write Policy

- Write through
 - easy to implement
 - eviction trivial
 - (just overwrite)
 - every write is slow (main memory time)
- Write back
 - fast (writes to cache)
 - eviction slow/complicate

Caltech CS184 Spring2005 -- DeHon

35

Cache Equation...

- Assume hits satisfied in 1 cycle
- $CPI = Base\ CPI + Refs/Instr (Miss\ Rate)(Miss\ Latency)$

Caltech CS184 Spring2005 -- DeHon

36

Cache Numbers

- $CPI = \text{Base CPI} + \text{Ref/Instr (Miss Rate)}(\text{Miss Latency})$
- From ch2/experience
 - load-stores make up ~30% of operations
- Miss rates
 - ...1-10%
- Main memory latencies
 - 50ns
- Cycle times
 - 300ps ... shrinking

Caltech CS184 Spring2005 -- DeHon

37

300ps Cycle
30ns Main mem

Cache Numbers

- No Cache
 - $CPI = \text{Base} + 0.3 * 100 = \text{Base} + 30$
- Cache at CPU Cycle (10% miss)
 - $CPI = \text{Base} + 0.3 * 0.1 * 100 = \text{Base} + 3$
- Cache at CPU Cycle (1% miss)
 - $CPI = \text{Base} + 0.3 * 0.01 * 100 = \text{Base} + 0.3$

Caltech CS184 Spring2005 -- DeHon

38

Wrapup

Caltech CS184 Spring2005 -- DeHon

39

Big Ideas

- Structure
 - temporal locality
- Model
 - optimization preserving model
 - simple model
 - sophisticated implementation
 - details hidden

Caltech CS184 Spring2005 -- DeHon

40

Big Ideas

- Balance competing factors
 - speed of cache vs. miss rate
- Getting best of both worlds
 - multi level
 - speed of small
 - capacity/density of large

Caltech CS184 Spring2005 -- DeHon

41