Memory Hierarchy

David A. Eckhardt
School of Computer Science
Carnegie Mellon University

de0u@andrew.cmu.edu

Outline

Lecture versus book

- Some of Chapter 2
- Some of Chapter 10

Memory hierarchy

- A principle (not just a collection of hacks)

Am I in the wrong class?

"Memory hierarchy": OS or Architecture?

- Yes

Why cover it here?

- OS manages several layers
  - RAM cache(s)
  - Virtual memory
  - File system buffer cache
- Learn core concept, apply as needed

You can't have it all

Memory Desiderata

- big
- fast
- cheap
- compact
- cold
- non-volatile (can remember w/o electricity)

Pick one

- ok, maybe two

Why?

- Bigger -> slower (speed of light)
- Bigger -> more defects (assuming constant per unit area)
- Faster, denser -> hotter (at least for FETs)

Users want it all

The ideal

- Infinitely large, fast, cheap memory
- Users want it (those pesky users!)
- They can't have it
    - Ok, so cheat!

Locality of reference

- Users don't *really* access 4 gigabytes uniformly by byte
- 80/20 "rule"
    - 80% of the time is spent in 20% of the code
    - Great, only 20% of the memory needs to be fast!

Deception strategy

- harness 2 (or more) kinds of memory together
- secretly move information among memory types

Cache

Small, fast memory...

- ...backed by a large, slow memory
- ...indexed according to the large memory's address space
- ...containing the most popular parts (now)

SRAM cache holds popular pixels

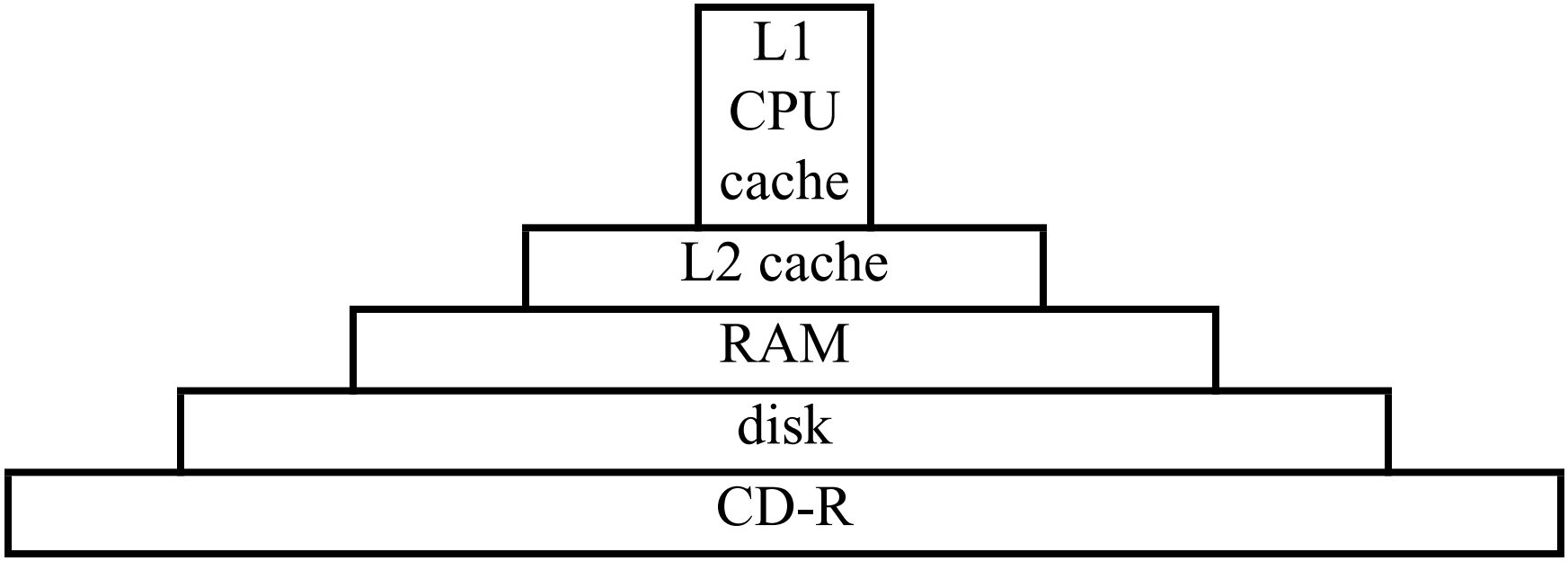
- DRAM holds popular image areas
  - Disk holds popular satellite images
    - Tape holds one orbit's worth of images

Clean general-purpose implementation?

- No: tradeoffs different at each level
  - size ratio: data address / data size
  - speed ratio
  - access time = f(address)

But the *idea* is general-purpose

Deception Picture



The questions

- Line size
- Placement/search
- Miss policy
- Eviction
- Write policy

Today's Examples

L1 CPU cache

- Smallest, fastest
- Maybe on the same die as the CPU
- Maybe 2nd chip of multi-chip module
- As far as CPU is concerned, this is *the* memory

Disk block cache

- Holds disk sectors in RAM
- Entirely defined by software
- You will implement one

Line size

"Line size" = item size

- Many caches handle fixed-size objects
  - Simpler
  - Predictable operation times

L1 cache line size

- 4 32-bit words (486, IIRC)

Disk cache line size

- Maybe disk sector (512 bytes)
- Maybe "file system block" (small # of sectors)

Picking a Line Size

What should it be?

- See “locality of reference”
  - (“typical” reference pattern)

Too big

- Waste throughput
  - Fetch a megabyte, use 1 byte
- Reduce “hit rate”
  - String move: *q++ = *p++
  - Better have at least two cache lines!

Too small

- Waste latency
  - Frequently need to fetch another line

Content-Addressable Memory

RAM

- store(address, value)
- fetch(address) -> value

CAM

- store(address, value)
- fetch(value) -> address
    - Are we having P2P yet?

“It’s always the last place you look”

- Not with a CAM!

Cool!

- But fast CAMs are small (speed of light)

Placement/search

Placement = "Where can we put ____?"

- "Direct mapped" - each item has *one* place
  - Think: hash function
- "Fully associative" - each item can be *any* place
  - Think: CAM

Direct Mapped

- Placement & search are trivial
- False collisions are common
  - String move: *q++ = *p++
  - Each iteration could be *two* cache misses!

Fully Associative

- No false collisions
- Cache size limited by CAM size

Sample choices

L1 cache

- Often direct mapped
- Sometimes 2-way associative
- Depends on phase of transistor

Disk block cache

- Fully associative
    - Open hash table = large variable-time CAM
    - Fine since “CAM” lookup time << disk seek time

Choosing associativity

- Trace-driven simulation
- Packaging constraints

Miss policy

Miss policy: {Read,Write} X {Allocate,Around}

- Allocate: miss -> allocate a slot
- Around: miss -> don't change cache state

L1 cache

- *Mostly* read-allocate, write-allocate
- But not for "uncacheable" memory
    - ...such as Ethernet card ring buffers
- "Memory system" provides "cacheable" bit
- Some CPUs have "write block" instructions

Disk block cache

- *Mostly* read-allocate, write-allocate
- What about reading (writing) a huge file?
- see (e.g.) madvise()

Eviction

“The steady state of disks is ‘full’”.

- Each placement requires an eviction
- Easy for direct-mapped caches
- Otherwise, policy is necessary

Ideal policy - consult an oracle!

- Evict whichever item won’t be used longest
- Useful only in simulation comparisons

Least-recently-used (LRU)

- LRU *may* be a reasonable approximation of Ideal
  - (“Past performance does not guarantee future results”)
- Or it may be the *worst* possible thing
  - Cache size: 4 (fully associative)
  - Reference pattern: 1, 2, 3, 4, 5, ...

Eviction

Random

- Pick a random item to evict
- Randomness protects against pathological cases

Could "Random" be good?

- What would it take?

L1 cache

- LRU is easy for 2-way associative!

Disk block cache

- Frequently LRU, frequently modified
  - "Prefer metadata"
  - Other hacks

Write policy

Assume a write hit (not write-around)

Write-through

- Store new value in cache
- Also store it through to next level
- Simple

Write-back

- Store new value in cache
- Store it to next level only on eviction
    - "Mandatory optimization": "dirty bit"
- May save *substantial* work

Write policy

L1 cache

- It depends
- May be write-through if next level is L2 cache

Disk block cache

- Write-back
- Popular mutations
    - Pre-emptive write-back if disk idle
    - Bound write-back delay (crashes happen)

Translation caches

Address mapping

- CPU presents virtual address (CS:EIP)
- Fetch segment descriptor from L1 cache (or not)
- Now fetch page table entry from L1 cache (or not)
- Now fetch the actual word from L1 cache (or not)

"Translation lookaside buffer" (TLB)

- Observe result of segmentation, virtual->physical mapping
- Key = virtual address
- Value = physical address

Challenges

Write-back failure

- Power failure?
  - Battery-backed RAM!
- Crash?
  - Maybe the old disk cache is ok after reboot?

Coherence

- What about *shared* caches?
  - Multiprocessor: 4 L1 caches share L2 cache
  - TLB: v->p all wrong after context switch
- What about non-participants?
  - I/O device does DMA
- Solutions
  - Snooping
  - Invalidation messages

Summary

Memory hierarchy has many layers

- Size: kilobytes through terabytes
- Access time: nanoseconds through minutes

Common questions, solutions

- Each instance is a little different
    - But there are lots of cookbook solutions