Memory Hierarchy

Goals

- High capacity
- Fast access
- Cheap to build

Can't have all three

- Registers: very low capacity, but fast
- Main memory (RAM): pretty large capacity, but slow

Memory Hierarchy

Goals

- High capacity
- Fast access
- Cheap to build

Can't have all three

- Registers: very low capacity, but fast
- Main memory (RAM): pretty large capacity, but slow

Another problem: rate of growth

- CPU performance increasing 2x every 1.5 years
- Memory performance increasing 5% per year

Solution: Caches

Compromise between capacity and speed

Store frequently/recently used data

- Much smaller capacity than RAM (KBs or MBs)
- More expensive to build, more physical space

Managed entirely by hardware

- Abstraction is that we have a large, fast memory

Analogy

Your desk

Limited space, easy access

Green Library

Lots of material, slow to access

Analogy

Your desk

Limited space, easy access

Your bookshelf

More space than your desk, faster than library

Green Library

Lots of material, slow to access

Analogy

Your desk

Limited space, easy access

Your bookshelf

More space than your desk, faster than library

Green Library

Lots of material, slow to access

"Auxiliary Library" in Livermore

Huge capacity, takes a day to page in materials

Why Caches?

Key assumption: locality

Memory accesses are not totally random

Temporal locality

Likely to access same piece of memory again

E.g. local variables

Spatial locality

Likely to access neighboring memory

E.g. contiguous elements of array

Caching Terms

Hit: data found in cache

Miss: data not found in cache

- Go to next level (another cache, RAM) to get it

Caching Terms

Hit: data found in cache

Miss: data not found in cache

Go to next level (another cache, RAM) to get it

Miss rate: fraction of accesses which miss

Hit time: if hit, time to access data

Miss penalty: time we have to wait if we miss

myth Memory Hierarchy

Registers: 16 * 8 bytes

- Can read/write two registers in 1 cycle

L1 cache: 32 KiB, 1-2 cycle hit time

- Very fast, pretty small
- Optimized for hits

L2 cache: 4-6 MiB, 5-20 cycle hit time

- Much larger than L1, but 10x slower
- Optimize to avoid misses

RAM: 4 GiB, 50-200 cycle access time

Effect of Miss Rate

Scenario: 1% miss rate

1 cycle hit time, 100 cycle miss penalty

Average memory access time (AMAT)

```
1 cyc * 99 hits + 100 cyc * 1 miss = 1.99 cyc/access
```

What if miss rate increased to 3%?

Effect of Miss Rate

Scenario: 1% miss rate

1 cycle hit time, 100 cycle miss penalty

Average memory access time (AMAT)

```
1 cyc * 99 hits + 100 cyc * 1 miss = 1.99 cyc/access
```

What if miss rate increased to 3%?

```
AMAT = 1 * 97 + 100 * 3 = 3.97 cyc/access
```

2x slower on average!

Cache Mechanics

Cache divided into blocks

- If miss, read whole block into cache
- Bigger block => bring in more neighbors

How to find block holding a given address?

- Check every block: very expensive in hardware
- Every address maps to one block: addresses could conflict

Cache Mechanics Example

64 byte cache block, 128 entries

- Each address maps to a single block

Address = `...xxx yyyyyyy zzzzzz`

- `...xxx`: tag (determine if block holds this address)
- `yyyyyyy`: which block in cache to look in
- `zzzzzz`: offset into block

Examples

Writing Cache-Friendly Code

Can't just avoid linked lists

- But consider whether another data structure might be better

If using linked list

- Put most recently used cells at front of list
- Consider the cost of accessing next cell
- Group accesses to the same cell

Measure, don't guess!

- Are cache misses the bottleneck?
- Are there misses where you don't expect?