23-cache-p2-notes

Caching: P2

Agenda:

- 0. Re-Orienting
- 1. Tradeoffs
- 2. Locality
- 3. The General Structure
- 4. Writing

Reading: ...on up to 6.5 and 6.6, too

slides.pdf (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/23-cache-p2/slides.pdf)

---

0. Re-Orienting

Computation...all the way down to the laws of physics.

Mapping high-level languages to ISAs

Building datapaths out of combo logic (with some registers too)

Then using sequential logic to get the datapath implementing the ISA

But what about making it **faster?**.

Two main techniques:

- Caching
- Pipelining

---

1. Tradeoffs

Which approach works best?

What blocksize is best?

---

2. Locality

Conflict miss

Capacity miss

Working set

The reason why it works!

Spatial locality

Temporal locality

Think about it in terms of:

- actual data
- example code
  - draw out address space
  - work through example touches

stride k

Demonstrating an extreme case: **demo.c (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/23-cache-p2/demo.c)**

Applying it: Practice Problem 6.9 and Figure 6.22

Reason...

Run... (**622.c (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/23-cache-p2/622.c)**)

Or get actual address traces (via valgrind), and then analyze them with a program that simulates the cache strategy!

E.g..... using N=1000....

- **clear1 (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/23-cache-p2/622-1.txt)**
- **(https://ssl.cs.dartmouth.edu/~sws/cs51-s15/23-cache-p2/622-1.txt) clear2 (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/23-cache-p2/622-2.txt)**
- **(https://ssl.cs.dartmouth.edu/~sws/cs51-s15/23-cache-p2/622-1.txt) clear3 (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/23-cache-p2/622-3.txt)**

3. The General Structure

Split caches.

Levels of caches.

Think about impact of relative miss costs!

Von Neumann and Harvard, again...

the Memory Pyramid

4. Writing

Write hits

- write-through
- write-back (dirty bit). when evicted.

Write misses

- write-allocate
- no-write-allocate

What's missing from this discussion?