Lecture #15

****
Review -- 1 min
****
Memory: Motivation and Overview

Year 2000 v 1980

memory:CPU performance ratio – 1000x worse (for memory)

→ Complex designs
→ limits ILP – long use after load delay

Cache Performance

CPUtime = IC * (CPI_exec + Misses Per Instruction * Miss Penalty) * Clock Cycle Time

- Miss Penalty is time *in addition to* hit time

****
Outline - 1 min
****
4 Questions of Memory Design

1. Block Placement
2. Block Identification
3. Block replacement
4. write strategy

3C's
3 Ways to Improve Cache Performance
XX Ways to Improve Miss Rate
(some of them)

Unifying idea: framework for reasoning about memory system design

****
Preview - 1 min
****
One of the comments – preview next class
Next Class:

reducing miss penalty
reducing hit time

****
Lecture - 20 min
****
4 Questions for Memory Hierarchy Designers

Q1: Block Placement: Where can a block be placed in the upper level?
Picture: fully associative, direct mapped, set associative

Q2: Block identification: How is a block found if it is in the upper level?

```
|                  Block Address            |  Block offset |
|       Tag                  |    index     |               |
```

PICTURE: cache -- indexed by index
select block with offset
compare with tag

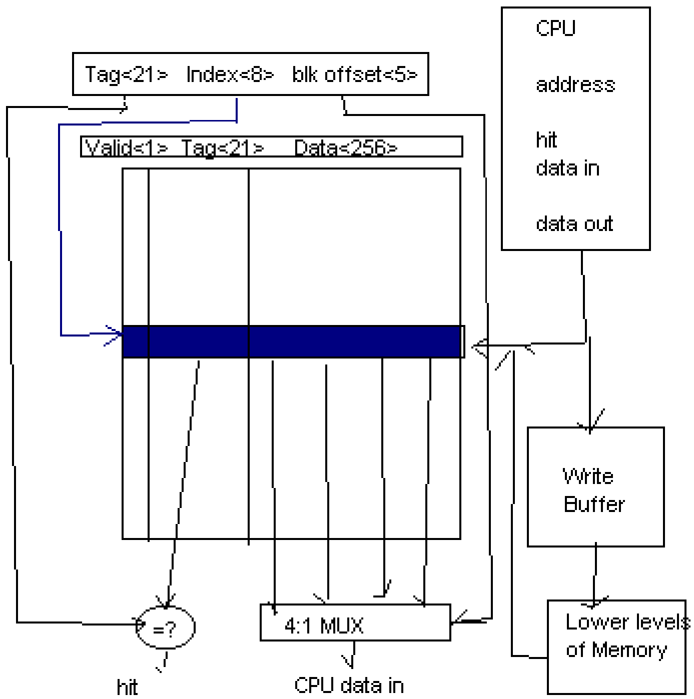
Q3: Block replacement: Which block should be replaced on a miss?
Easy for direct mapped (only 1 choice)

Otherwise several choices:
- Optimal: replace block used furthest into future
- LRU -- works well , but lots of HW
- Random works pretty well
- Random but not MRU often a good compromise

Q4: Write Strategy: What hapens on a write?
write back
write through
(w or w/o write buffer)

Example: Alpha 21064 Data Cache

8192K Byte
Direct Mapped
block = 4 x 64-bit words
--> 256 blocks (8K / 32 bytes per line)



Improving Performance

Cache Performance

CPUtime = IC * (CPI_exec + Misses Per Instruction * Miss Penalty)
* Clock Cycle Time
Miss Penalty is time *in addition to* hit time

3 parts to equation
1. Reduce the miss rate
2. reduce the miss penalty
3. reduce time to hit in cache

Reducing the miss rate

Classifying misses: the 3C's
1. Compulsory
    #misses in infinite cache
    first access to a block must be a miss

    AKA: cold start misses, first reference misses

QUESTION: What determines the # of cold start misses
to run a program?
(e.g. a simple formula)
    data set size / block size

2. Capacity
    #misses in fully associative cache of size X - # misses in inf cache

    happens when blocks are discarded and then retrieved

3. Conflict

    misses in n-way associative cache - # misses in fully assoc

    AKA collision, interference misses

(4. Coherence)
    sometimes add a 4th C

    example: multiprocessor
    (draw picture)

        proc1: read A
        proc2: write A
        proc 1: readA

Most obvious ways to reduce misses

1. increase cache size
2. increase associativity

SLIDES: miss rate v. cache size – fig 5.10

Observations:

1. Compulsory not too important

2. C's are not independent
increasing size reduces conflicts
--> a miss's clssification depends on particular parameters
(one miss can sometimes be thought of as conflict,
sometimes as capacity)

This is a problem with the model

3. 2:1 cache rule holds
(a rule of thumb)

miss rate of 1-way cache of size X
~= miss rate of 2-way cache of size X/2

SLIDE: fraction of miss rate – fig 5.10 part 2

Observations

1. as size increases conflict and compulsory
become more important

QUESTION: If you were building caches, why is it important
to know how many compulsory misses there are?
(or to divide misses into three types?)
A: Tell's you where to spend your effort.
e.g. if compulsory misses were a big fraction, spending $'s

on larger caches won't help much.

Lesson here: measure before building. Remember amdahl's law.

example: building disk cache for web
    Fully associative --> no conflict misses
    Disks are cheap to make big --> no capacity misses
        (e.g. DEC had a 3GB shared cache that ran for
        months w/o discarding something for capacity)
    2 sources of misses
        1. compulsory/cold start
            someone touches data they've never touched before
            ZIPF distribution
        2. consistency/communication
            read a page that has changed

--> don't spend your time working on replacement algorithms
or ways to build bigger disk caches

****
Admin - 3 min
****
HW3 due week from Friday
More papers to read for friday
Project proposals due today

****
Lecture - 24 min
****

Ways to Reduce Miss Rate

APPROACH 1: Bigger caches
    Advantage: Reduce Capacity Misses
    DA: o more transistors = costs more

o larger = slower

QUESTION: are we hitting diminishing returns?

APPROACH 2: Increase Associativity

Reducing Misses via Higher Associativity

(see previous slides)

Q: Which C does this help?

Adv: Reduce conflict misses

2:1 cache rule -- miss rate 1-way assoc cache size X
= miss rate 2-way assoc size X/2

---> conflict misses are significant

DA: May slow access time
(Remember -- TIME is true measure of performance not hit rate)

e.g. Hill [1988] hit time for external cache +10% (+2% for internal)
for 2-way v. 1-way

memory access time may increase even if hit rate falls

SLIDE: Avg Memory Access TIme v. Miss Rate (fig 5.14)

WORSE: may hurt cycle time
--> not only are memory accesses slowed, all instructions are slowed
--> tend to use direct mapped for level-1 cache

Dilemma: Want associativity (2:1 rule suggests lots of conflict misses)
Don't want slower cache

Several attempts to get best of both worlds

Associativity Hack 1: Victem Cache

Goal: Fast hit time of direct mapped while avoding conflict misses

Approach:
Buffer for data last discarded from cache

Picture: figure 5.15
regular cache with small buffer hanging off of it

Result:
Jouppi [1990]: 4-entry victem cache removed 20-95% of conflicts for 4KB direct mapped data cache

Real-life Examples:
used in Alpha, HP machines

Associativity Hack 2: Pseudo Associativity
Goal:
Combine fast hit time of direct while avoiding conflict

Approach: Probe cache twice
(serialize 2-way associative cache)
hit first probe -- as fast as direct cache
hit 2nd probe -- small penalty

Difficulties

1. want first probe to hit to be better than 2-way
    (otherwise get 50% hit rate on first probe
    --probably wouldn't be better than slight
    slowdown in access time on first prob

--> solution -- LRU between entries (swap on 2nd probe hit)

2. Complicates CPU pipeline
(variable hit time)
--> used in L2 cache

Real-world examples
    MIPS R10000 L2 Cache
    similar ideas in Ultrasparc cache (predicts which set youll hit)

NEXT APPROACH (APPROACH 3)

Question: Suppose we change the block size? Which
of 3C's affected?
    --> compulsory
    (2nd order effects: conflict, capacity)

SLIDE: Miss Rate v. block size – fig 5.11

APPROACH 3: Prefetching (and increased block size)

Larger block is simple example
    works b/c after touch byte 1 of block
    expect to touch byte 8 (spatial locality)

More sophisticated: prefetch multiple blocks

Hardware Instruction Prefetching
----
example -- Alpha 21064 fetches 2 blocks on a miss

Draw Picture:

CPU

Cache

Memory

When cache goes to memory, fetch next block too.
(Into ***stream buffer)***
CHeck stream bufer on miss

Jouppi [1990] -
1 data stream bufer gets 25% of misses
4 streams gets 43%

Palacharia & Kessler [1994] -- scientific programs
8 streams: 50-70%
(for 4-way 64KB cache)

QUESTION: Which of 4C's does prefetching reduce?
A: compulsory -- yes
A: capacity, conflict -- yes (b.c. reduces misses on later accesses)

QUESTION: What problems can prefetching cause?
1. increase BW requirements
(fetch more blocks -- some prefetched blocks won't be used
general trade-off -- prefetching increases bandwidth demands
to improve latency
(often a good choice -- spend $'s to get BW, hard to reduce
latency -- can't bribe God)

2. hurt hit rate

prefetched data competes with data already in cache
(saw that in large block picture)

Software Prefetching

Explicit data prefetch instructions
~load but don't affect registers and can't cause exception

e.g. HP PA-RISC, MIPS IV, PowerPC, SPARCv9

limitation -- have to issue prefetch instructions (may delay other instructions)
is cost of prefetch issue < savings from reduced misses

Tech trends: superscaler reduces cost of issue
memory gets more expensive

--> expect to see more of this

APPROACH 4: REDUCING MISSES VIA COMPILER OPTIMIZATIONS

Instructions
reorder procedures in memory to reduce conflict misses
profiling to find conflicts

McFarling [1989] -- reduce cache misses by 75% on 8KB direct mapped cache with 4 byte blocks

Side note: this makes doing experiments hard
    small change in order you link procedures
    can be 25%-50% change in performance
    --> make sure speedup comes from where
    you think it does

Data

Merging arrays:
    improve spatial locality w. single array of compound elements
    v. 2 single arrays
    example

```
/* Before */
int val[size];
int key[size];

/* after */
struct record{
        int val;
        int key;
};
struct record merged[size];
```

    /* this is better sw engineering, too */

Loop Interchange
    change nesting of loops to access data in order stored in
    memory

    e.g. if stored in column-major order
    PICTURE: square of data -- want to access top to bottom and
        then across, not left to right and then down

Loop Fusion
    combine 2 independent loops that have same variables
    and some variables overlap

    e.g.

```
loop{
        calculate a[]
}
loop{
        calculate b[] = a[] * x
}

/* better */
loop{
        calculate a[];
        calculate b[] = a[] * x;
}
```

Blocking

improve temporal locality by accessing "blocks" of data repeatedly v. going down whole columns or rows

side note: invented by I/O programmers to avoid disk accesses

Example:

successive relaxation
e.g. you've got a piece of metal with a flame and an ice cube on it and want to find how the temperatures at diferent points change over time

(each point gets average of neighbors at each time step)

naive -- column major order
better -- recursive or blocked.

SLIDE: impact of compiler improvements
(Lebeck and Wood hand-optimization of programs
("inspiration to compiler writers")

- table 1 and figure 4 from lebeck.ps

- “Cache profiling and the SPEC benchmarks: a case study” Lebeck and Wood

****
Summay - 1 min
****

3 ways to improve cache performance
1. reduce misses
2. reduce miss time
3. improve hit time

4 approaches to reducing misses
1. bigger caches
2. more associativity
3. prefetching
4. compiler tricks

Next time:
finish reducing misses
look at reducing miss time and improving hit time