Lecture #16

****
Review -- 1 min
****
3C's -- Compulsory, Capacity, Conflict (and Coherence)
3 Ways to improve cache performance
    Reduce Miss Rate
    Reduce Miss Penalty
    Reduce Hit Time
4 Approaches to Improve Miss Rate

****
Outline - 1 min
****
Improving Miss Rate
1. Bigger Caches
2. Increase Associativity
3. Prefetching
4. Compiler Optimizations

****
Preview - 1 min
****
M: Reduce Miss Penalty
W: Improving Hit Time, main memory
F: paper discussion
M: main memory, Case Study -- VM, Alpha 21064 Memory Hierarchy
W: Review/Exam

****
Lecture - 20 min
****

Ways to Reduce Miss Rate

(from last lecture)
APPROACH 1: Bigger caches
    Advantage: Reduce Capacity Misses
    DA: o more transistors = costs more
        o larger = slower

APPROACH 2: Increase Associativity

Reducing Misses via Higher Associativity

    (see previous slides)
(did this last lecture – skip)

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

        SKIP: SLIDE: Avg Memory Access TIme v. Miss Rate

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: Victim Cache

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

Approach:
Buffer for data last discarded from cache

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

idea: associativity is only needed in uncommon case when 2 popular items hash to same bucket; don't slow down common case for uncommon case

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)

****
Admin - 3 min
****
discussion of feedback

hand out reading for Friday (vote: do it Monday instead?)

HW

****

Lecture - 24 min
****

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 buffer 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 porofiling and the SPEC benchmarks: a case study" Lebeck and Wood

****
Summay - 1 min
****
Improving Miss Rate

1. Bigger Caches
2. Increase Associativity
3. Prefetching
4. Compiler Optimizations