Lecture #17

****
Review -- 1 min
****
Improving Miss Rate

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

****
Outline - 1 min
****
5 ways to Reduce Miss Penalty
Ways to improve hit time

****
Preview - 1 min
****

F: Main Memory/Case Study -- VM, Alpha 21064 Memory Hiearchy
M: Discuss papers
W: Review/Exam

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

5 Ways to Reduce Miss Penalty
----
1. Read Priority over Write on a miss
2. Subblock Placement
3. Early Restart and Critical Word First
4. Nonblocking Caches
5. 2^{nd} Level Caches

1. Read Priority over Write on a miss (SKIP)

Intuition:

reads are blocking

writes are nonblocking

→ do reads first → CPU stalls less

Approach:

CPU writes go to write buffer

Challenges:

RAW hazards

Solution: check write buffer before allowing read to proceed

2. Subblock Placement; 3. Early Restart; Critical Word First

Motivation:

cache block > word/double read by CPU

→ don't wait for whole line

Approaches:

subblocks – just fetch part of the block

(need valid bits for different parts of block)

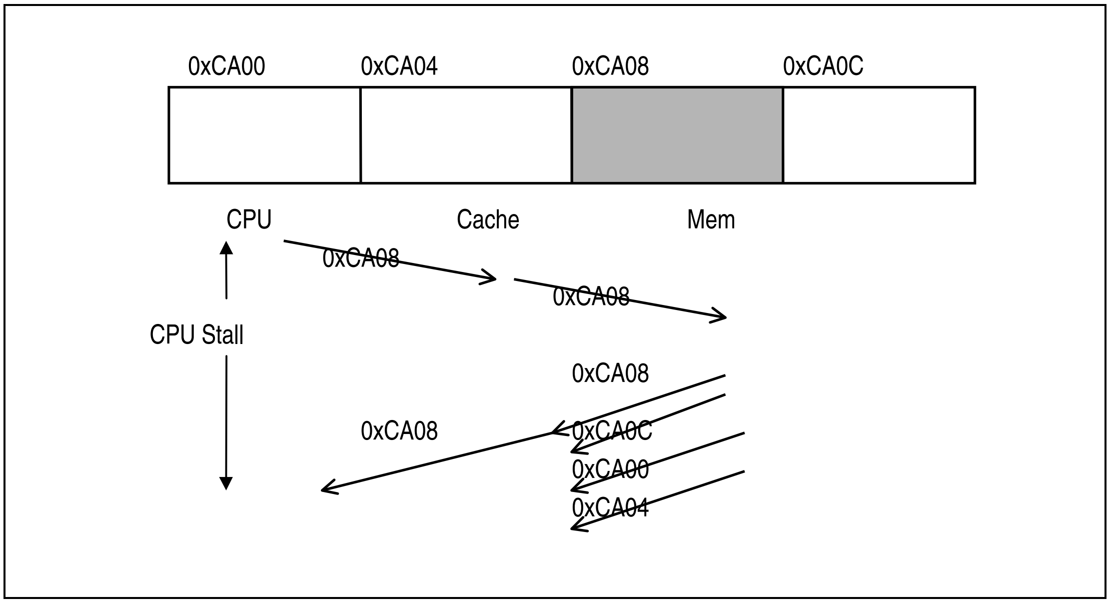
early restart – restart CPU as soon as requested word arrives

critical word first – Cache requests desired byte from lower level

lower level supplies whole line, starting

from requested byte and then wrapping

e.g. (critical word first + early restart)



Limitations:

- most useful for large blocks
- tech trends: mem BW improving faster than mem latency
- spatial locality: tend to want next word/whole block

4. Non-blocking Caches

Idea: allow requests to memory system while a miss being serviced

2 varieties

1. Hit under miss – satisfy cache hits during a miss
2. hit under multiple miss or miss under miss – overlap multiple misses

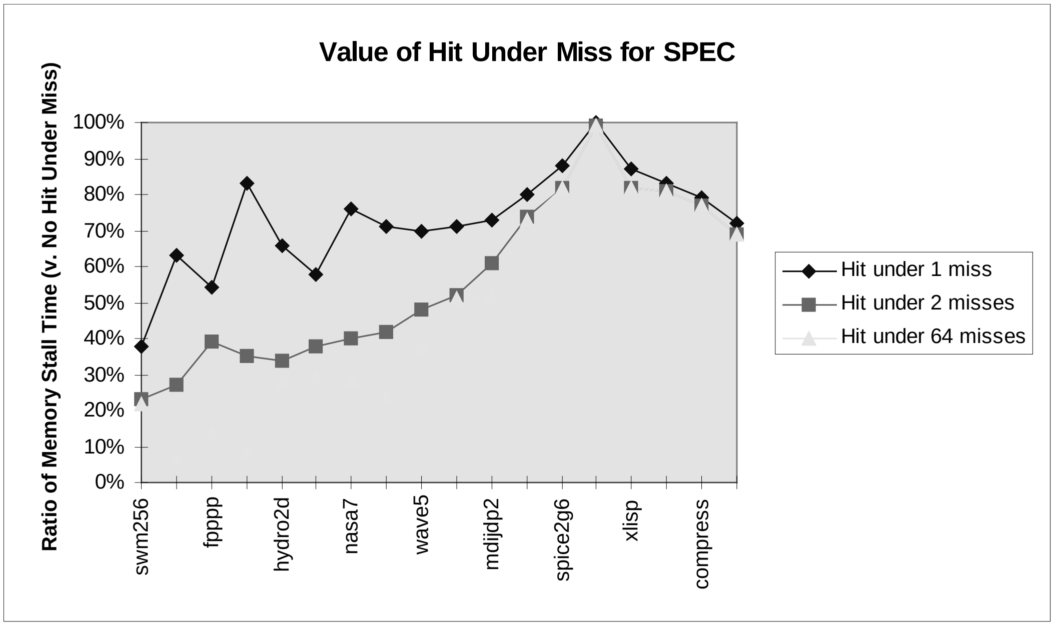
Only useful if allow out-of-order execution

QUESTION: why

Limitation:

- significantly increases complexity of cache controller/memory subsystem

Can be big win: (Farkas and Jouppi 1994)



FP Avg: AMAT = 0.68 (base)
0.52 (HU 1 miss)
0.34 (HU 2 misses)
0.26 (HU 64)
Int Avg: AMAT = 0.24 (base)
0.20 (HU 1)
0.19 (HU 2)
0.19 (HU 64)

5. Second Level Cache

Idea: Go a cache for a cache – go to L2 cache instead of main memory
Picture:
<draw picture>

L2 Equations:
AMAT = Hit Time_L1 + MissRate_L1 * Miss Penalty_L1

QUESTION: What is Miss_Penalty_L1?
Miss Penalty_L1 = Hit Time_L2 + Miss Rate_L2 * Miss Penalty_L2

QUESTION: Is Miss_Rate_L2 the Local or Global Miss Rate?

A: local

→ global miss rate_L2 = Miss Rate_L1 * Miss Rate_L2

L2 Cache Design

We talked about 4 ways to reduce cache misses

- bigger cache
- more associativity (+ assoc hacks)
- prefetching (hw, sw)
- compiler optimizations

QUESTION: Which techniques apply to L2?

Recursive: multi-level cache systems

Current state of the art:

3 levels of caches (e.g. Alpha)

QUESTION: Why not have 10 levels of cache?

Ideas: 1) costs %

2) costs time (increases time to access DRAM)

****

Admin - 3 min

****

Exam:

don't worry too much – 15% of grade (2.5 homeworks)

goal is **not** to create a grade distribution

goal: teaching tool – get you to spend some time thinking about stuff

format

3 hours next Wednesday night

design exam to take 1.5 hours

closed book
you may bring 1 sheet of 8.5x11 paper
write anything on it you want

here's what I'm planning --
questions will not be rote repetition
combine or extend ideas you've seen
test understanding

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

Next set of techniques

Improving hit time:

Take home message:

1. cache hit time sets cycle time
2. Not so much reading the data as deciding you've got the *right* data

ways to improve hit time

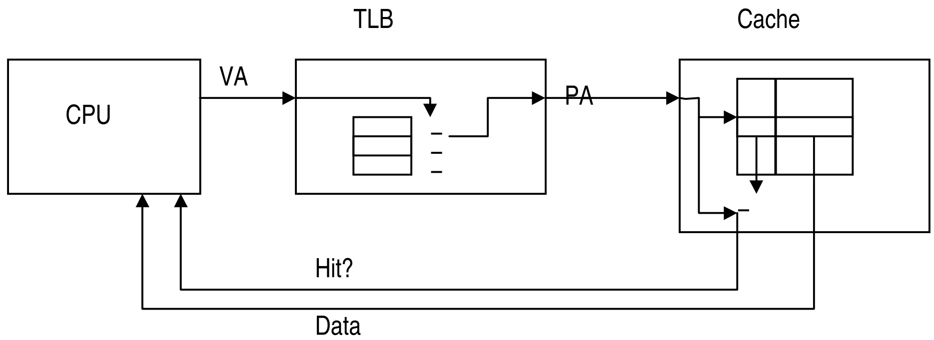
1. small, simple caches
   e.g. direct mapped → transmit data before/during tag check

2. pipeline read hits
   e.g. MIPS R4000 (a few weeks ago)
   READ DATA TAG

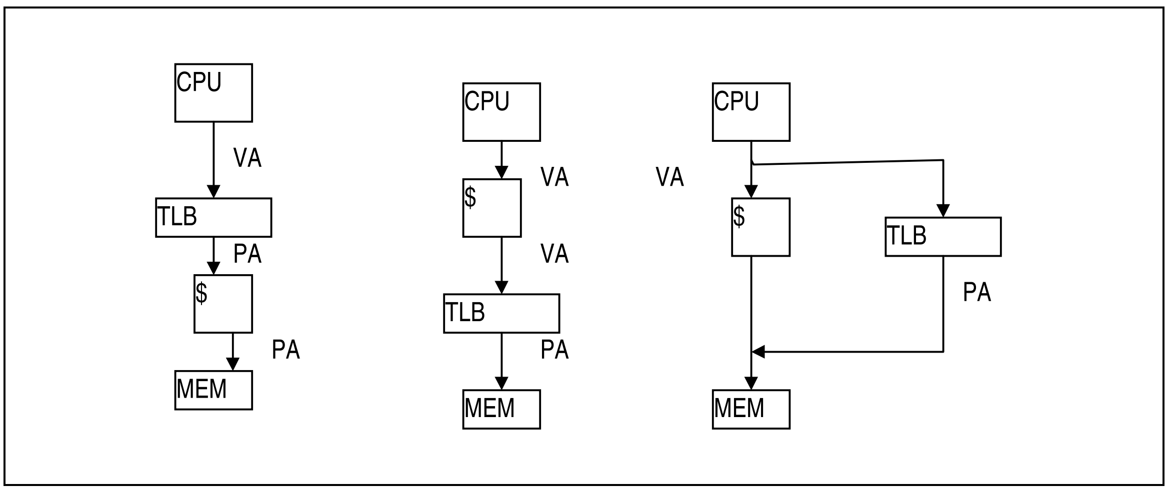
3. pipeline write hits
   check tag *before* writing data (for obvious reasons)
   → writes have long "latency", but don't care as long as we can initiate one write per cycle
   (need to check reads against write buffer)

4. avoid address translation

CPU uses *Virtual Addresses*
Mem System uses Physical Addresses
TLB Translates from VA to PA



Traditional: 2 serial cache lookups on critical path!



Approach 2: Virtually Addressed Cache
2 problems

1. aliases
   solutions
   1. page coloring
   → page coloring – if 2 VA's map to same PA then lower N bits will match
   → both addresses map to same cache line

   (Still get cache miss for one of the addresses, but better than getting wrong answer on a write)

2. virtually addressed, physically tagged cache
   (pipleline:
   1. start read cache[VA]; start PA = TLB(VA)
   2. get data and physical tag; check physical tag against PA

   (This is picture 3 above)
   QUESTION: What is DA?
   ANSWER: cache index < page size

   → sometimes use set associative to get around this
   e.g. IBM 3033 – 16-way set assoc to allow 64KB physically tagged cache with 4K pages.

2. cache flush on process switch
   problem: different processes use same VA for different things
   naïve solution: flush cache on proc. Switch (hurts performance)
   → add process identifier tag – only a hit if a match

QUESTION: how would you implement process identifier tag
   (main ideas – set a PID register in CPU on context switch
   design goal—minimize bits of storage → use few bits
   to cover common case of a couple processes running at same time (flush cache if recycle PID))

****
Summay - 1 min
****
5 Techniques for reducing miss cost
1. Read priority over write
2. subblock placement
3. Early restart, crit word first
4. nonblocking cahces
5. second level caches

Techniques for improving hit time