Computer Systems Organization

CSE 521/560M
Lecture 16
Prof. Patrick Crowley

Plan for Today

- Questions
- Today’s discussion
Improving Cache Performance

- CPI contributed by cache
  - Miss rate * number of cycles to handle the miss
- Execution time contributed by cache
  - Clock cycle time * CPI
  - Clock cycle time might depend on cache organization
- To improve cache performance
  - Decrease miss rate without increasing time to handle the miss
  - Decrease time to handle the miss (i.e., penalty) without increasing miss rate
- In general, larger caches require larger access times

Obvious Solutions to Decrease Miss Rate

- Increase cache capacity
  - Larger means slower
  - Limitations for on-chip caches
- Increase cache associativity
  - Diminishing returns (perhaps after 4-way)
  - More comparisons (i.e., more logic)
  - Make cache look more associative than it really is
What About Cache Block Size?

- For a given application, cache capacity and associativity, there is an optimal block size
- Benefits of long cache blocks
  - good for spatial locality (code, vectors)
  - reduce compulsory misses
- Drawbacks
  - increase access time
  - Increase inter-level transfer time
  - Fewer blocks imply more conflicts

Miss Rate vs. Block Size

© 2003 Elsevier Science (USA). All rights reserved.
Example Block Analysis

- We are to choose between 32B and 64B blocks
  - With miss rates $m_{32}$ and $m_{64}$, resp.
- Miss penalty components
  - Send request + access time + transfer time
  - Example
    - Send request (block independent): 2 cycles
    - Access time (assume block ind.): 28 cycles
    - Transfer time depends on bus width. Say our bus is 64 bits wide, then transfer time is twice as much for 64B (8 cycles) than for 32B (4 cycles)
  - 32B is better if $34 \times m_{32} < 38 \times m_{64}$

Example (cont’d)

- From the text (p. 427):
  - For a 16K cache $m_{32}=2.87$, $m_{64}=2.64$
  - For a 64K cache $m_{32}=1.35$, $m_{64}=1.06$
- Thus, 32B is better for 16K and 64B better for 64K
Associativity

- The same kind of analysis can be used for associativity (miss rate vs. cycle time)
- “Old” conventional wisdom
  - Direct-mapped caches are faster; cache access is bottleneck for on-chip L1; make L1 direct-mapped
  - For on-board (L2) caches, direct-mapped are 10% faster
- “New” conventional wisdom
  - Can make 2-way set-associative cache fast enough for L1. Allows larger caches to be addressed only with page offset bits
  - Looks like it does not make much difference for L2/L3 caches

Victim Cache: Bringing Associativity to a Direct-Mapped Cache

- Goal: remove some of the conflict misses
- Idea: place a small, fully-associative buffer “behind” the L1 cache and “before” the L2 cache
- Procedure
  - L1 hit, do nothing
  - L1 miss in block b, hit in victim v, pay extra cycle, swap b and v
  - L1 miss in b, victim miss, add b to victim (evict if necessary)
- Victim buffer of 4 to 8 entries in a 4KB cache works well
Victim Cache

Improving Hit Rate via Prefetching

- Goal: bring data in cache *just in time* for its use
  - Not too early, otherwise might displace other useful blocks (i.e., *pollution*)
  - Not too late otherwise will have *hit-wait* cycles
- Constrained by (among others)
  - Imprecise knowledge of instruction stream
  - Imprecise knowledge of data stream
Why, What, When & Where

- Why
  - Hide memory latency, reduce cache misses
- What
  - Ideally, a semantic object
  - In practice, a cache line or sequence of them
- When
  - Ideally, just in time
  - In practice, depends dynamically on prefetching technique
- Where
  - Either the cache or a prefetch buffer

How

- Hardware prefetching
  - *Nextline* prefetching for instructions (many processors bring in a block on a miss and the following one as well)
  - OBL “one block lookahead” for data prefetching (many variations, can be directed by previous successes)
  - Stream buffers [Jouppi, ISCA 90] is an example that works well for instructions but not for data
  - Stream buffers with hardware *stride* prediction mechanisms
    - Works well for scientific programs with disciplined data access patterns
How (cont’d)

- Software prefetching
  - Use of special instructions (cache hints: touch in PowerPC, load in register 31 for Alpha)
  - *Non-binding* prefetch (prefetch ignored if an exception occurs)
  - Must be inserted by software (usually via compiler analysis)
  - Advantage: no special hardware needed
  - Drawback: increases instruction count

Compiler and Algorithm Optimizations

- Tiling or blocking
  - Reuse contents of the data cache as much as possible (great for linear algebra programs)
- Same type of compiler analysis as for parallelizing code
- Code reordering
  - Particularly for improving Icache performance
  - Can be guided by profile information
Reduce Cache Miss Penalty

• Give priority to reads (e.g., write buffers)
• Send the requested word first
  – Critical word or wrap around strategy
• Lock-up free (i.e., non-blocking) caches
• Cache hierarchy

Write Buffers

• Reads are more important than
  – Writes to memory if WT cache
  – Replacement of dirty lines in WB
• Hence buffer the writes in write buffers
  – FIFO queues to store data temporarily (loads must check contents of buffer)
  – Since writes have a tendency to come in bunches, write buffer must be deep
  – Allow read miss to bypass the writes in the write buffer (when there are no conflicts)
Coalescing Write Buffers and Write Caches

- **Coalescing (or merging) write buffers**: writes to an address (block) already in the write buffer are combined
- Extend write buffers to small fully associative write caches with WB strategy and dirty bit. Natural extension for write-through caches.

Lock-up Free Caches

- Proposed in the early 1980’s but implemented only recently due to complexity
- Idea: allow cache to have several outstanding miss requests simultaneously (i.e., *hit under miss*)
- Single hit under miss (a la HP PA1700) relatively simple
- For several outstanding misses, require the use of MSHR’s (miss status holding registers)
MSHR’s

- Each MSHR should hold
  - A valid (busy) bit
  - Address of the requested cache block
  - Index into the cache where the block will go
  - Comparator (to prevent using the same MSHR for a miss to the same block)
  - If data is to be forwarded to CPU at the same time as the cache, needs register name
  - Implemented in MIPS R10K, Alpha 21164, Pentium Pro

Cache Hierarchy

- Two and even three levels of caches are quite common now
- L2 is very large, but L1 filters many hits so “local” hit rate can appear low
- L2, in general, has longer blocks and higher associativity
- Multi-level inclusion property is implemented most of the time (also with L1 WI and L2 WB)
Assignment

• Readings
  – For Wednesday
    • H&P: 5.3-5.5
  – For Monday
    • H&P: 5.6-5.7