

Caches (Ch 7 of P&H)

Reconciling CPU/Memory Speeds

- Memory access in a pipelined processor
 - do the VA to PA translation
 - access memory
- CPU cycle time versus memory access time?
- What happens to the pipeline whenever there is a memory ref instruction?
- How to bridge the gap between processor speed and memory access latency?
 - problem only because of virtual memory?

Memory Hierarchy

- fast is expensive, slow is cheap
- less of fast, more of slow
- three levels
 - main memory
 - secondary cache (also called L2)
 - primary cache (also called L1)
- what is a cache?
- what is the intent?
- why does this hierarchy work?

Principles of Locality

- spatial
- temporal
- hits and misses
 - servicing a miss
 - penalty for a miss
 - effective memory cycle time
- inclusion property
- caches introduced in 60's
- today: every machine contains a cache

Basics

- Organization
- How do we know a data item is in cache?
- If it is, how do we find it?
- Three things: location, lookup, validity
 - location
 - e.g. direct-mapped
 - mapping function: (memory address) modulo (cache size)
 - lookup
 - if a cache entry contains data, what memory location does it correspond to?
 - cold start

Example

- El Cheapo machine
 - 8 word cache(direct mapped),16 word memory
- initially empty
 - consider the following references (all reads)
 - 0, 1, 2, 3, 1, 3, 0, 8, 9, 10 (decimal addresses)
 - what happens when we access address 8?
- Interpreting the memory address
 - <tag, index>
 - e.g. 32-bit address for a 64Kbyte cache with a word size of 32 bits (4 bytes)
 - hardware?

Handling Misses

- Return to our pipelined design
 - IF and MEM states
 - where do the instructions and data come from?
 - I-cache and D-cache
 - hits are easy
 - how do we handle misses in the control?
 - consider IF state
 - what should we do?
 - same treatment for MEM state as well
 - can we cheat on misses?
 - IF state
 - MEM state: stall-on-use

Basic Cache Algorithm

- Read
 - send address to cache (either from PC or ALUout register)
 - if cache signals hit life is peachy; else send address to memory; when data comes back write it into cache; supply it to the CPU
 - in a pipelined control also send NOPS (bubbles) down the pipeline so that CPU is busy doing nothing!

- Writes

- two choices

- write-through (DECstation 3100)

- write to cache (no need to check hit or miss)

- change the cache tag if necessary

- write to main memory

- continue when done

- » upshot? Performance

- » solution: write buffer

- write-back

- write only to cache (no need to check hit or miss)

- change the cache tag if necessary

- when do we update main memory?

To combine or Not?

- Miss rate is an important metric of cache performance, but....
 - have to know which metric to use when....
- I-cache and D-cache
 - pros of combining
 - reduced miss rate
 - cons of combining
 - reduced bandwidth (i.e. contention when IF and MEM stages need to access the cache simultaneously)
 - which is more important?

Improving Cache Performance

- CPU time
 - $(\text{CPU exec. cycles} + \text{Memory-stall cycles}) * \text{clock}$
 - memory stall cycles
 - $(\text{instr. per program}) * (\text{misses per instr.}) * \text{miss-penalty}$
 - may have to account differently if read and write misses cost different penalties
 - ***work out examples in Sec 7.3 of text***
- reducing memory stalls
 - reduce miss rate
 - blocksize, flexible placement
 - reduce miss penalty
 - wide memory

Spatial Locality

- Multiword cache block
 - consider 32 bit address, 64Kbyte cache, 4 bytes per word, and 4 words per cache block
 - handling reads
 - simultaneous access of all words during tag comparison
 - mux to select the particular word on hit
 - handling writes
 - any difference from single word cache block?
 - Performance with larger block size?

Flexible Placement

- Fully associative
 - memory block can be placed anywhere in cache
 - lookup?
 - search all cache blocks
 - $\langle \text{tag}, \text{block-offset} \rangle$
- Set associative
 - memory block can be placed anywhere within a set of cache blocks
 - lookup?
 - search all blocks within a set of cache blocks
 - $\langle \text{tag}, \text{index}, \text{block-offset} \rangle$

- Extremes of set associativity
 - partition cache into “p” parallel caches
 - » this is called p-way set associative
 - » “p” parallel hardware for access
 - direct-mapped
 - » 1-way set associative
 - » n sets (each containing 1 block)
 - » hardware is simple
 - fully-associative
 - » n-way set associative (where n = total number of blocks in the entire cache)
 - » 1 set (containing n blocks)
 - » hardware is complex
 - practical considerations
 - 8-way set-associative as good as fully-associative

Replacement

- Direct mapped
 - no choice
- Set- or Fully-associative
 - LRU
 - what does it involve?

Putting it all together

- L1 - 1 cycle; L2 - 4 cycle; M - 10 cycles
- CPU generates VA in IF or MEM states
- translate VA to PA
 - TLB helps but lookup in the critical path...
- use PA to look up primary cache (I or D)
 - hit
 - pipeline continues without a hiccup...
 - miss
 - access secondary cache
 - hit: bubbles in the pipeline for next 4 cycles
 - miss bubbles in the pipeline for next 10 cycles

Getting Translation out of Critical Path

- Do cache lookup and translation in parallel
 - how is that possible?
 - Consider portion of VA unchanged in translation
 - use unchanged portion of address to index cache
 - after cache access use the translated PA to validate cache access
 - any restriction?
 - Other ideas:
 - virtually indexed physically tagged
 - virtual tagged caches
 - lot more on caches in follow-on course....

Memory System Design

- Consider the following
 - address to memory: 1 cycle
 - DRAM access: 15 cycles
 - data transfer: 1 cycle
- Memory and bus-width of 1 word
 - access time: $1 + 15 * 4 + 4$
- Memory and bus-width to match blocksize
 - access time: $1 + 15 + 1$
- Interleaved memory design (wide memory but smaller bus width)
 - access time: $1 + 15 + 4$