## Cache Performance

- CPI $_{\text{contributed by cache}}$ = $\text{CPI}_c$
  = miss rate * number of cycles to handle the miss
- Another important metric
  Average memory access time = cache hit time * hit rate
  + Miss penalty * (1 - hit rate)

## Improving Cache Performance

- To improve cache performance:
  - Decrease miss rate without increasing time to handle the miss (more precisely: without increasing average memory access time)
  - Decrease time to handle the miss w/o increasing miss rate
- A slew of techniques: hardware and/or software
  - Increase capacity, associativity etc.
  - Hardware assists (victim caches, write buffers etc.)
  - Tolerating memory latency: Prefetching (hardware and software), lock-up free caches
  - O.S. interaction: mapping of virtual pages to decrease cache conflicts
  - Compiler interactions: code and data placement; tiling

## Obvious Solutions to Decrease Miss Rate

- Increase cache capacity
  - Yes, but the larger the cache, the slower the access time
  - Limitations for first-level (L1) on-chip caches
  - Solution: Cache hierarchies (even on-chip)
  - Increasing L2 capacity can be detrimental on multiprocessor systems because of increase in coherence misses
- Increase cache associativity
  - Yes, but "law of diminishing returns" (after 4-way for small caches; not sure of the limit for large caches)
  - More comparisons needed, i.e., more logic and therefore longer time to check for hit/miss?
  - Make cache look more associative than it really is (see later)

## What about Cache Block Size?

- For a given application, cache capacity and associativity, there is an optimal cache block size
- Long cache blocks
  - Good for spatial locality (code, vectors)
  - Reduce compulsory misses (implicit prefetching)
  - But takes more time to bring from next level of memory hierarchy (can be compensated by "critical word first" and subblocks)
  - Increase possibility of fragmentation (only fraction of the block is used – or reused)
  - Increase possibility of false-sharing in multiprocessor systems

## What about Cache Block Size? (c'ed)

- In general, the larger the cache, the longer the best block size (e.g., 32 or 64 bytes for on-chip, 64, 128 or even 256 bytes for large off-chip caches)
- Longer block sizes in I-caches
  - Sequentiality of code
  - Matching with the IF unit

## Example of (naïve) Analysis

- Choice between 32B (miss rate *m32*) and 64B block sizes (*m64*)
- Time of a miss:
  - send request + access time + transfer time
  - send request independent of block size (e.g., 4 cycles)
  - access time can be considered independent of block size (memory interleaving) (e.g., 28 cycles)
  - transfer time depends on bus width. For a bus width of say 64 bits transfer time is twice as much for 64B (say 16 cycles) than for 32B (8 cycles).
  - In this example, 32B is better if $(4+28+8)m32 < (4+28+16)\,m64$

## Example of (naïve) Analysis (c'ed)

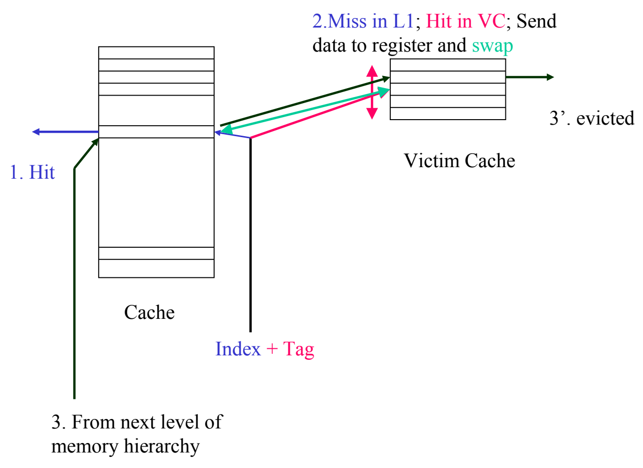
- Case 1. 16 KB cache: *m32* = 2.87, *m64* = 2.64
  - 2.87 * 40 < 2.64 * 48
- Case 2. 64KB cache: *m32* = 1.35, *m64* = 1.06
  - 1.35 * 40 > 1.06 * 48
- 32B better for 16KB and 64B better for 64KB
  - (Of course the example was designed to support the "in general the larger the cache, the longer the best block size " statement of two slides ago).

## Impact of Associativity

- "Old" conventional wisdom
  - Direct-mapped caches are faster; cache access is bottleneck for on-chip L1; make L1 caches direct mapped
  - For on-board (L2) caches, direct-mapped are 10% faster.
- "New" conventional wisdom
  - Can make 2-way set-associative caches fast enough for L1. Allows larger caches to be addressed only with page offset bits (see later)
  - Looks like time-wise it does not make much difference for L2/L3 caches, hence provide more associativity (but if caches are extremely large there might not be much benefit)

## Reducing Cache Misses with more "Associativity" -- Victim caches

- First example (in this course) of an "hardware assist"
- Victim cache: Small fully-associative buffer "behind" the L1 cache and "before" the L2 cache
- Of course can also exist "behind" L2 and "before" main memory
- Main goal: remove some of the conflict misses in L1 direct-mapped caches (or any cache with low associativity)


2.Miss in L1; Hit in VC; Send data to register and swap
3'. evicted
Victim Cache
1. Hit
Cache
Index + Tag
3. From next level of memory hierarchy

## Operation of a Victim Cache

- 1. Hit in L1; Nothing else needed
- 2. Miss in L1 for block at location *b*, hit in victim cache at location *v*: swap contents of *b* and *v* (takes an extra cycle)
- 3. Miss in L1, miss in victim cache : load missing item from next level and put in L1; put entry replaced in L1 in victim cache; if victim cache is full, evict one of its entries.
- Victim buffer of 4 to 8 entries for a 32KB direct-mapped cache works well.

## Bringing more Associativity -- Column-associative Caches

- Split (conceptually) direct-mapped cache into two halves
- Probe first half according to index. On hit proceed normally
- On miss, probe 2nd half ; If hit, send to register and swap with entry in first half (takes an extra cycle)
- On miss (on both halves) go to next level, load in 2nd half and swap

## Skewed-associative Caches

- Have different mappings for the two (or more) banks of the set-associative cache
- First mapping conventional; second one “dispersing” the addresses (XOR a few bits)
- Hit ratio of 2-way skewed as good as 4-way conventional.

## Reducing Conflicts --Page Coloring

- Interaction of the O.S. with the hardware
- In caches where the cache size > page size * associativity, bits of the physical address (besides the page offset) are needed for the index.
- On a page fault, the O.S. selects a mapping such that it tries to minimize conflicts in the cache .

## Options for Page Coloring

- Option 1: It assumes that the process faulting is using the whole cache
  - Attempts to map the page such that the cache will access data as if it were by virtual addresses
- Option 2: do the same thing but hash with bits of the PID (process identification number)
  - Reduce inter-process conflicts (e.g., prevent pages corresponding to stacks of various processes to map to the same area in the cache)
- Implemented by keeping “bins” of free pages