

Software Prefetching

- Use of special instructions (cache hints: **touch** in Power PC, **load in register 31** for Alpha, **prefetch** in Intel micros)
- **Non-binding** prefetch (in contrast with proposals to prefetch in registers).
 - If an exception occurs, the prefetch is ignored.
- Must be inserted by software (compiler analysis)
- Advantage: no special hardware
- Drawback: more instructions executed.

Metrics for Prefetching

- **Coverage:** Useful prefetches/ number of misses without prefetching
- **Accuracy:** useful prefetches/ number of prefetches
- **Timeliness:** Related to number of hit-wait prefetches
- In addition, the usefulness of prefetching is related to how critical the prefetched data was

Techniques to Reduce Cache Miss Penalty

- Give priority to reads -> Write buffers
- Send the requested word first -> critical word or wrap around strategy
- Sectored (subblock) caches
- Lock-up free (non-blocking) caches
- Cache hierarchy

Write Buffers

- Reads are more important than:
 - Writes to memory if WT cache
 - Replacement of dirty lines if WB
- Hence buffer the writes in **write buffers**
 - Write buffers = FIFO queues to store data
 - Since writes have a tendency to come in bunches (e.g., on procedure calls, context-switches etc), write buffers must be *“deep”*

Write Buffers (c'ed)

- Writes from write buffer to next level of the memory hierarchy can proceed in parallel with computation
- Now loads must check the contents of the write buffer; also more complex for cache coherency in multiprocessors
 - Allow read misses to bypass the writes in the write buffer

Critical Word First

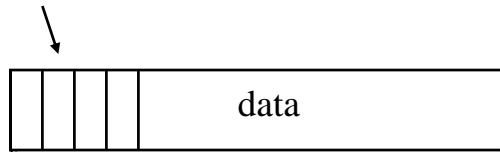
- Send first, from next level in memory hierarchy, the word for which there was a miss
- Send that word directly to CPU register (or IF buffer if it's an I-cache miss) as soon as it arrives
- Need a one line buffer to hold the incoming line (and shift it) before storing it in the cache

Sectored (or subblock) Caches

- First cache ever (IBM 360/85 in late 60's) was a sector cache
 - On a cache miss, send only a subblock, change the tag and invalidate all other subblocks
 - Saves on memory bandwidth
- Reduces number of tags but requires good spatial locality in application
- Requires status bits (valid, dirty) per subblock
- Might reduce false-sharing in multiprocessors
 - But requires metadata status bits for each subblock
 - Alpha 21164 L2 uses a dirty bit/16 B for a 64B block size

Sector Cache

Status bits, in particular valid bit



tag	subblock1			subblockn

Lock-up Free Caches

- Proposed in early 1980's but implemented only within the last 15 years because quite complex
- Allow cache to have several outstanding miss requests (**hit under miss**).
 - Cache miss “happens” during EX stage, i.e., longer (unpredictable) latency
 - Important not to slow down operations that don't depend on results of the load
- Single hit under miss (HP PA 1700) relatively simple
- For several outstanding misses, require the use of MSHR's (**Miss Status Holding Register**).

MSHR's

- The outstanding misses do not necessarily come back in the order they were detected
 - For example, miss 1 can percolate from L1 to main memory while miss 2 can be resolved at the L2 level
- Each MSHR must hold information about the particular miss it will handle such as:
 - Info. relative to its placement in the cache
 - Info. relative to the “missing” item (word, byte) and where to forward it (CPU register)

Implementation of MSHR's

- Quite a variety of alternatives
 - MIPS 10000, Alpha 21164, Pentium Pro, III and 4
- One particular way of doing it:
 - Valid (busy) bit (limited number of MSHR's – structural hazard)
 - Address of the requested cache block
 - Index in the cache where the block will go
 - Comparator (to prevent using the same MSHR for a miss to the same block)
 - If data to be forwarded to CPU at the same time as in the cache, needs addresses of registers (one per possible word/byte)
 - Valid bits (for writes)

Cache Hierarchy

- Two, and even three, levels of caches in most systems
- L2 (or L3, i.e., board-level) very large but since L1 filters many references, “local” hit rate might appear low (maybe 50%) (compulsory misses still happen)
- In general L2 have longer cache blocks and larger associativity
- In general L2 caches are write-back, write allocate

Characteristics of Cache Hierarchy

- **Multi-Level inclusion (MLI)** property between off-board cache (L2 or L3) and on-chip cache(s) (L1 and maybe L2)
 - L2 contents must be a superset of L1 contents (or at least have room to store these contents if L1 is write-back)
 - If L1 and L2 are on chip, they could be mutually exclusive (and inclusion will be with L3)
 - MLI very important for cache coherence in multiprocessor systems (shields the on-chip caches from unnecessary interference)
- Prefetching at L2 level is an interesting challenge (made easier if L2 tags are kept on-chip)

Impact of Branch Prediction on Caches

- If we are on predicted path and:
 - An I-cache miss occurs, what should we do: stall or fetch?
 - A D-cache miss occurs, what should we do: stall or fetch?
- If we fetch and we are on the right path, it's a win
- If we fetch and are on the wrong path, it is not necessarily a loss
 - Could be a form of prefetching (if branch was mispredicted, there is a good chance that that path will be taken later)
 - However, the channel between the cache and higher-level of hierarchy is occupied while something more pressing could be waiting for it