AMAT: average memory access time

AMAT = Hit Time + MissRate × MissPenalty

## Reduce Hit Time

1. Reduce cache size (miss rate ↑)
2. Reduce cache associativity (miss rate↑)
3. Overlap one hit with other

Pipelined (L1) Caches (multiple cycles to access, dividing access to multiple stages):

1. Address index
2. Read tags and valid bits of each block in set
3. Data read
4. TLB and cache hit (virtual address -> TLB (frame number) -> physical address)

Instead, use virtual address to address data (no TLB latency), but:

1. TLB have permission (need to access for permission check) -> no advantage, still need to access TLB) …… 17.6
2. Virtual address is specific for one process -> need perform cache flush every context switch (expensive) …… 17.6

Combine virtually indexed and physically tagged cache:

Virtual Accessed Cache:

* A virtual address: tag + virtual index + cache offset, use virtual index to access data, also, parallel use TLB get tag and check whether hit (no need to worry for context switch, since tag is check) -> no need to flush cache, but still need to flush TLB (TLB flush penalty is small, comparing with cache flush. But write-back policy is time-consuming)

图片包含 文字, 人员

描述已自动生成

* Aliasing problem: (multiple virtual index to same physical address, thus, might same data, same physical address, in different virtual cache map to different address) -> every time write will require check for same data -> expensive: since aliasing problem is serious, we can’t use virtually accessed cache, although it reduce hit latency

Virtual Index physical tagged Cache

* Aliasing problem: lower bits in virtual address become page offset, and upper bits become page number (page number -> translate to frame number in physical address): use page number as tag, use page offset as index: if we use those physical address to index, there is no aliasing problem. -> cache need to be small

图片包含 文字

描述已自动生成

图片包含 人员

描述已自动生成

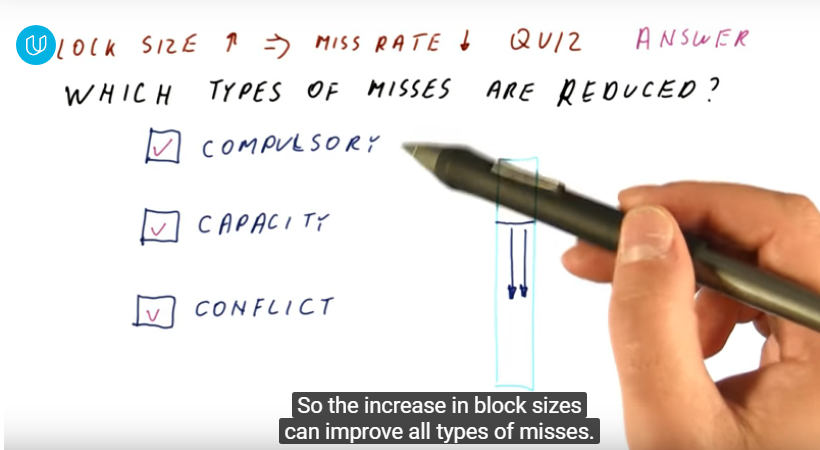
* Real VIPT caches: cache size <= assoc \* page size

## Reduce miss rate

1. Way prediction (predict which line in set is most likely to hit, if no hit there, perform normal set-associative cache) -> reduce miss rate while maintain hit time
2. Replacement policy:
   * NMRU replacement (not most recently use, approximate LRU)

Track which are the most recently used -> pick randomly chose other blocks to replace: for each set, need MRU pointer to tell which block is recently accessed (1 bit for 2-way associative, 2 bits for 4-way-associative)

* + PLRU (pseudo-LRU): 0 at initialization, one bit/line in set, every time line access, set bit to 1, when replace, pick 0; if all are 1, detect for that, once change the last 0 to 1, reset all other from 1 to 0. -> not reset in one-by-one manner, but bulk-reset (清零)

1. Larger caches blocks (miss rate ↓when spatial locality is good)
2. Pre-fetching (guess which blocks will be accessed soon, bring them into cache ahead of time) -> possible cons: cache pollution (bring not-used blocks)
   * Software pre-fetch: add pre-fetch instructions
   * Hardware pre-fetch: stream buffer (sequential), stride prefetcher (check whether data access address have some certain distance), correlation prefetcher (record fetch relation for access dequence)
3. Loop interchange

图片包含 文字

描述已自动生成

## Reduce miss penalty

* Overlap multiple misses (blocking cache (once miss, block until load) -> unblocking cache): when load request, continue proceeding (might encounter another load), sending multiple request to memory (multiple load can be completed once the first load complete)



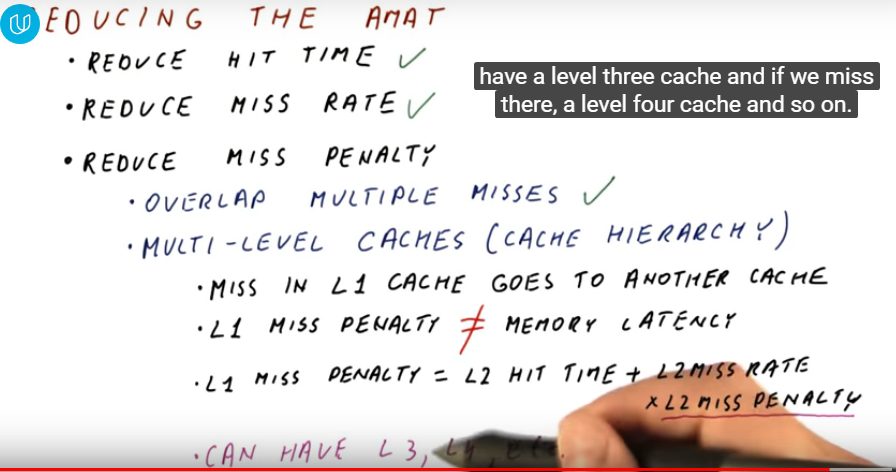
* + Miss under miss (miss status handling registers (MSHR)):

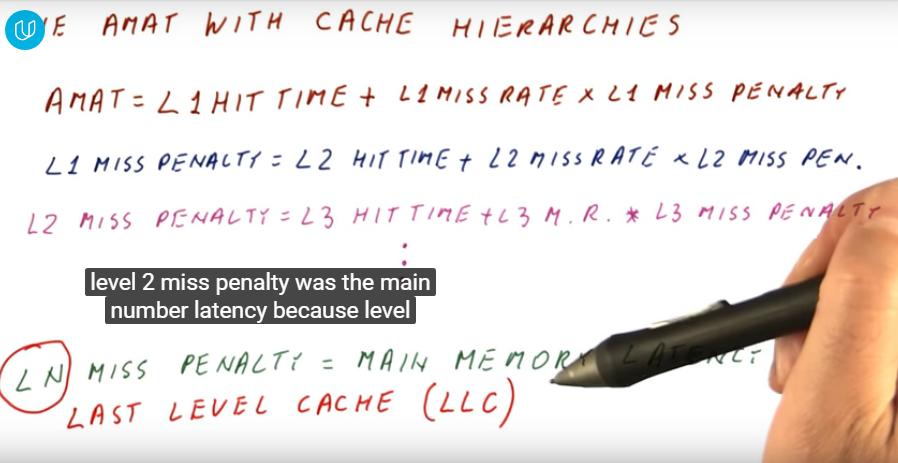
Record ongoing miss, if no previous miss match: allocate new block for miss; if previous miss match (not exactly the same word, but same block): half miss (if one by one blocking cache, will not happen), add that instruction to MSHR, when data block back, weak up all instruction in that MSHR

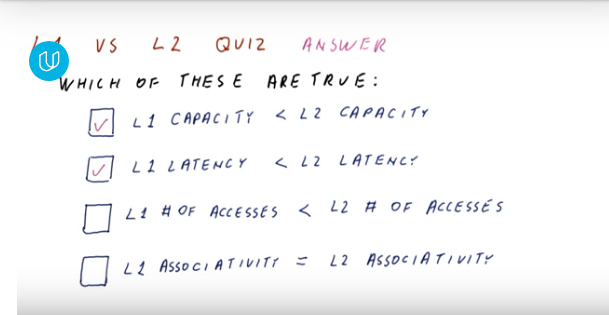
图片包含 人员

描述已自动生成

* Multi-level caches







Hit Rate

Local hit rate (percentage of all the access to that cache is hit)

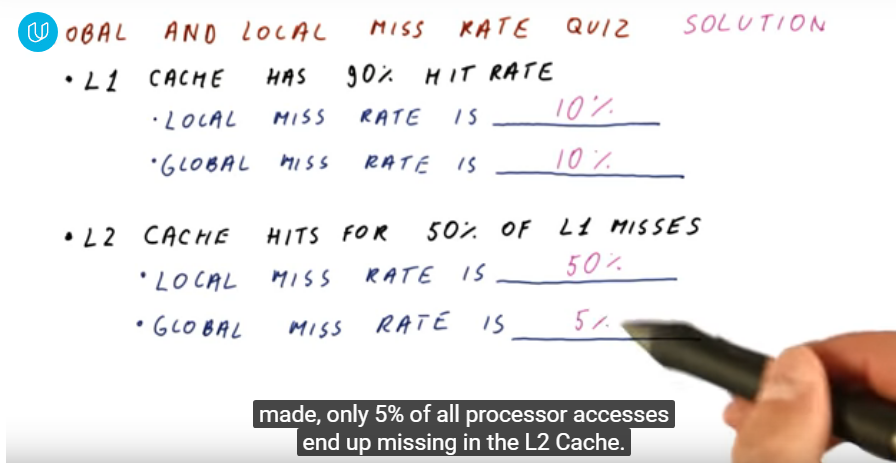
For example: L1 access will not go to L2 if hit, L2 only concern about local hit rate

* Global hit rate: 1- global miss rate

Global miss rate: # miss in this cache/ # all memory references

Local hit rate: # hits/ # access to this cache

Misses Per 1000 Instructions (MPKI)



* Inclusion property

Block in L1 might or might not in L2 (conflicting and replacement policy), need additional inclusion bit to check whether L2 block is in L1 to maintain the block in L2 (pros from write policy, for write back, inclusion ensures hit in L2 for L1 dirty block)