Advanced Caches

Improving the performance of caches is done by improving the hit rate, improving the miss rate, or improving the miss penalty. The methods to implement these improvements in discussed in this lesson.

Improving Cache Performance

Three methods for improving cache performance can be derived from the AMAT equation:

AMAT = Hit time + Miss Rate * Miss Penalty

Improvement Methods:

- Reduce Hit Time
- Reduce Miss Rate
- Reduce Miss Penalty

Reduce Hit Time

- Overlap cache hit with another hit
- Overlap cache hit with TLB hit
- Optimize Lookup for Common Case
- Maintain Replacement State Quicker

Pipelined Caches

Pipelining caches = Overlapping a cache hit with another hit
Hit Time = Actual Hit + Wait Time
To pipeline a cache:

- Partition the task into 3 stages:
  - Stage 1 - Reading the index to find the set
  - Stage 2 - Determining the hits and beginning the data read
  - Stage 3 - Finishing the data read

TLB and Cache Hit

Cache Speed-up Method 2: Overlap the cache hit with a TLB hit.
A cache that uses the physical address is called a physically indexed-physically tagged cache (PIPT cache).

Virtually Accessed Cache

If the virtual address is used by the cache, then the TLB and the cache can both complete their tasks at the same time.
The downside of the virtually addressed cache is the virtual address is process specific. So the cache needs to be flushed with every context switch, leading to cache misses.
A second downside would be aliasing - leading to incorrect execution.

Virtually Indexed - Physically Tagged Cache

Start with the virtual address. Partition into the offset, index, and tag.

- Index is used by the cache
- Tag is used get the frame number

The tag check is used with the translated physical address.
Both the cache lookup and the translation are done at the same time. It can be done without having to flush the cache and without aliasing if the cache is small enough.

VIPT Cache Aliasing

Virtual Address → Physical Address
VA Page Number = PA Frame #
VA Page Offset = LSB of PA
There is no aliasing if all the index bits come from the page offset. The cache has to be small enough to do this.

Real VIPT Caches

Cache size must be less than or equal to the Associativity of the cache * Page Size

Associativity and Hit Time

With higher associativity:

- Improves the miss rate but makes the hit time longer

With Direct Mapped:

- Improves the hit time but makes the miss rate worse

Way Prediction

With way prediction, the processor tries to guess which set is most likely to be a hit. If the prediction is correct the hit time goes down. If the prediction is wrong, then the normal set-associative check is used.

Way Prediction Performance

Using Way prediction the AMAT is better than with a direct mapped or a 8-way set associative cache.
Fully associative, 8-way set associative, and 2-way set associative can all benefit from way prediction.
Direct mapped caches do not benefit from way prediction.

Replacement Policy and Hit Time

Random selection has a good hit time, but poor miss rate.
LRU has a good miss rate, but requires updates of all counters on a hit - slowing the hit time and increases power consumption.

NMRU Replacement Policy
NMRU = Not-Most Recently Used
To get a better replacement policy than LRU or Random use NMRU.
Just track the MRU and replace any other block.

PLRU Replacement Policy
PLRU - Pseudo LRU
Every time a bit is accessed, set the LRU bit to 1. Kick out the LRU bit that equals 0. When all the LRU bits are 1, set the LRU block to 1 and reset all the LRU bits to 0.

PLRU is a compromise between LRU and NMLRU.

Reducing the AMAT
AMAT can be reduced by a lower hit time and a lower miss rate.
Causes of Misses:
- 3 Cs:
  - Compulsory Miss - when a block is accessed for the first time
  - Capacity Miss - blocks evicted when the cache is full
  - Conflict Miss - blocks are evicted due to associativity

Larger Cache - reduces capacity misses
Larger Associativity - reduces conflict and some capacity misses

Larger Cache Blocks
Larger cache blocks bring more words in
- The miss rate improves with good spatial locality
- The miss rate degrades with poor spatial locality

Increased block size can reduce compulsory, capacity, and conflict misses.

Prefetching
Prefetch blocks into the cache to improve the miss rate.
Good guesses eliminate misses, while bad guesses cause cache pollution.

Prefetching Instructions
The compiler can be used to determine which blocks to fetch.
The question becomes - how far in advance should the blocks be fetched?
This is a difficult question to answer with a compiler.

Hardware Prefetching
Hardware prefetching = using hardware to guess what will be accessed soon.
Types of Hardware prefetching:
- Stream buffer - prefetch the next block

Stride prefetch - prefetch the block at distance ‘d’
Correlating prefetcher - if A is fetched, then B should be prefetched, etc.

Loop Interchange

Loop interchange = compiler optimizations to change the code to have better locality by swapping the inner and outer loops to sequentially access the matrix in memory.

Overlap Misses

Reducing the Miss Penalty can be accomplished by overlapping the misses.

While the OOO processor is waiting for a cache miss, it continues to execute instructions. Within these instructions, the processor may encounter another cache miss. This is an overlapping miss.

Blocking Cache: Only one load at a time can be performed. If a cache gets a load instruction, no other load instruction can be executed until the first load is completed.

Non-blocking Cache: a non-blocking cache will allow:

Hit-under-miss: if the processor is waiting for a miss, it can execute cache hits.
Miss-under-miss: if the processor is waiting for a miss, it can execute additional requests to memory (other misses).

The non-blocking cache can reduce the performance penalty for a miss by half, this is memory level parallelism.

Miss Under Miss Support in Caches

Miss Status Handling Registers (MSHR):

-keep information about misses that are in progress
-check MSHRs to see if the requested cache miss is one that has already been requested.

if the cache miss is a new miss (a true miss):

1. allocate an MSHR register
2. track which instruction is waiting for the miss

if the cache miss has already been requested (a Half-miss):

1. the instruction is added to the MSHR

When data arrives from memory:

-The MSHR is used to alert the correct instructions that their requested data is ready.
-Release the MSHR register

How many MSHR registers are necessary?

16 - 32 MSHR is a typical number

Applications that can benefit from the MSHR are those that have misses often enough so the next miss can be issued and executed before the processor runs out of resources while "stalling" on the previous miss.

Cache Hierarchies

Different level caches are used to reduce the AMAT

If there is a miss in the first level cache, a second level cache is checked. If this level is a hit, then the time spent going to main memory is saved.

L1 miss penalty = L2 hit time + L2 miss rate * L2 miss penalty

AMAT with Cache Hierarchies

AMAT = L1 hit time + L1 miss rate*L1 miss penalty
L1 miss penalty = L2 hit time + L2 miss rate * L2 miss penalty
L2 miss penalty = L3 hit time + L3 miss rate * L3 miss penalty

The equations will continue until the L# miss penalty = the main memory latency. This is called the Last Level Cache (LLC)
L1 capacity < L2 capacity
L1 latency < L2 latency

Multi-Level Cache Performance

Having a cache is better than no cache, and having a 2 level cache is better than having a one level cache.

Hit Rate

Local hit = a hit to an individual cache. For example a local hit for the L2 cache is a hit in the L2 cache.
Global hit = the hits to all the caches.

Global Vs Local Hit Rate

Global Hit Rate = 1 - Global Miss Rate

Global Miss Rate = # of Misses in this cache / # of all memory accesses

Local Hit Rate = # of Hits / # of Access to this cache

MPKI = Misses per 1000 instructions

Inclusion Property

There are 3 different possibilities for the same block being in L1 and L2:

1. A block in L1 may or may not be in L2
2. A block in L1 must also be in L2 (Inclusion)
3. A block in L1 cannot also be in L2 (Exclusion)

If inclusion or exclusion is not enforced, then the block may or may not be in L2

To enforce inclusion an inclusion bit is required. The bit will track if a block is in the other level caches.

Aliasing in Virtually Accessed Caches

Aliasing is a big problem in virtually addressed caches.
Aliasing will result in incorrect execution. So every write to the cache must check for aliasing, this will lead to a degradation in performance.