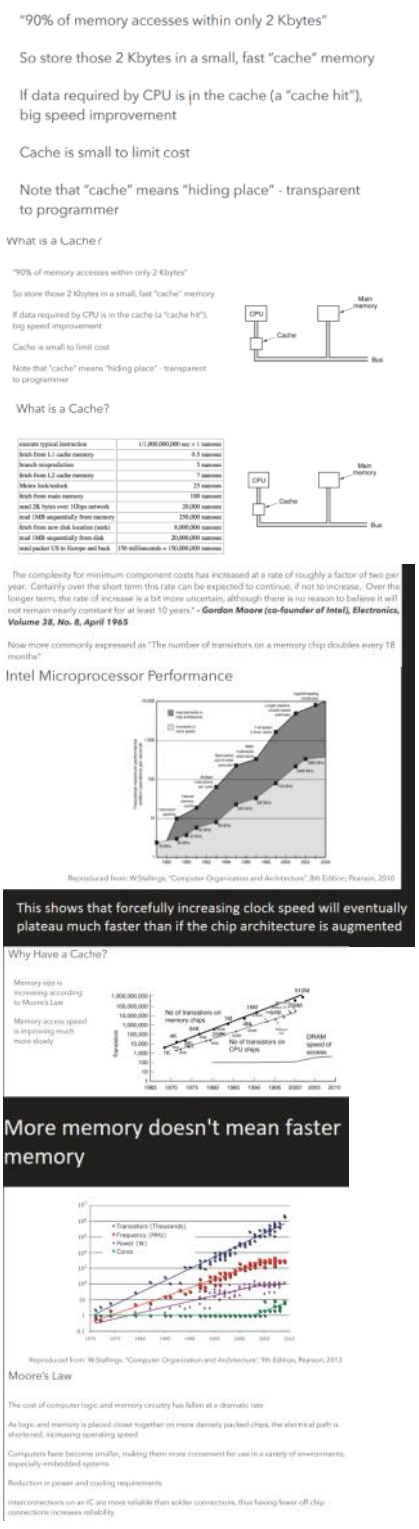
Memory Systems (Cache)

08 March 2025 00:43



This shows that forcefully increasing clock speed will eventually plateau much faster than if the chip architecture is augmented

1) Compulsory Misses

- *a. Occur the first time a block of data is accessed and it is not in the cache (e.g., cold start).*
- *b. These misses are inevitable and do not depend on the size of the cache.*

2) Capacity Misses

a. Happen when the cache is too small to hold all the necessary blocks during program execution.
b. Even with an optimal placement strategy, this type of miss occurs due to insufficient cache size.

3) Conflict Misses

a. Result from the cache's placement strategy.
b. For example, if the cache is not fully associative (e.g., direct-mapped), some blocks might conflict with each other and be evicted even when the cache has empty space.

4) Coherency Misses

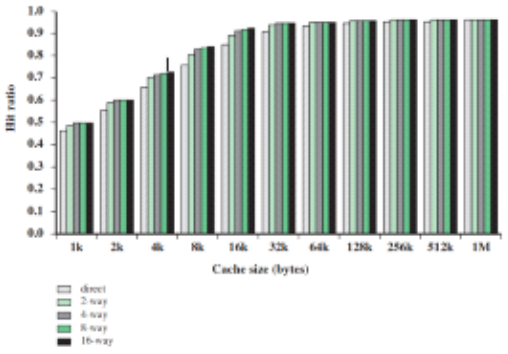
a. Occur in multiprocessor systems due to issues with maintaining consistency across caches (e.g., one processor's cache has outdated data compared to another processor's cache).

1. **Cache Associativity and "Ways" Explained**
   a. **What does "ways" mean in cache associativity?**
   b. When we talk about **"ways"** in cache, we're describing how **flexible** the cache is when storing data. The **total size of the cache stays the same**, but the organization of the cache changes.
   c. **Key Idea: Associativity Controls Flexibility, Not Size**
      i. A 1-way cache (direct-mapped):
      Each memory block can only be stored in 1 specific slot in the cache.
      ii. A 16-way cache:
      Each memory block can be stored in any of 16 slots within its set. This reduces the chances of conflicts (cache misses), where two blocks compete for the same slot.
   d. **How Does Associativity Work?**
      i. **Direct-mapped (1-way):**
         1) Each block of memory is mapped to exactly 1 slot in the cache.
         2) If multiple memory blocks need the same slot, one gets kicked out, causing a miss.
      ii. **16-way associative:**
         1) The cache is divided into sets, and each set has 16 slots (lines).
         2) A memory block can be stored in any of the 16 slots in its set.
         3) This flexibility reduces conflicts, improving the hit ratio (more efficient cache use).
2. **Analogy: Bookshelves**
   a. Imagine you have a **bookshelf** (the cache) with **16 slots** (cache lines):
   b. **Direct-mapped (1-way):**
      i. Each book (memory block) has a specific slot where it must go.
      ii. If two books are assigned to the same slot, one book gets kicked out, causing a conflict (miss).
   c. **16-way associative:**

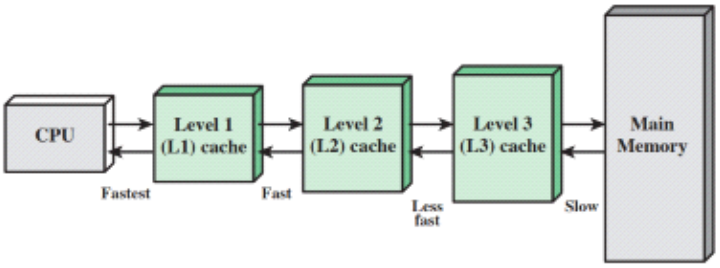
i. *The bookshelf is divided into sets of 16 slots.*
ii. *A book can go into any of the 16 slots in its set.*
iii. *This reduces conflicts because there are more places to store each book.*

3. ***Why Does Associativity Matter?***
   a. ***More ways (e.g., 16-way):***
      i. *Fewer conflicts, higher hit ratio, more flexibility.*
      ii. *Tradeoff: More complex hardware, slightly slower access due to searching multiple slots.*
   b. ***Fewer ways (e.g., 1-way):***
      i. *Simpler, faster lookup, but more conflicts, leading to a lower hit ratio.*
4. ***Key Takeaways:***
   a. *Higher associativity gives each memory block more options for storage, reducing misses.*
   b. *The cache does not get larger—it's just more flexible in how it organizes the same amount of storage.*
   c. *The tradeoff is complexity in the hardware to search all the slots in a set.*

L1 Cache Size and Complexity



Multilevel Cache



Relating the Memory Hierarchy to Cache

Data storage is a balance of size and speed:

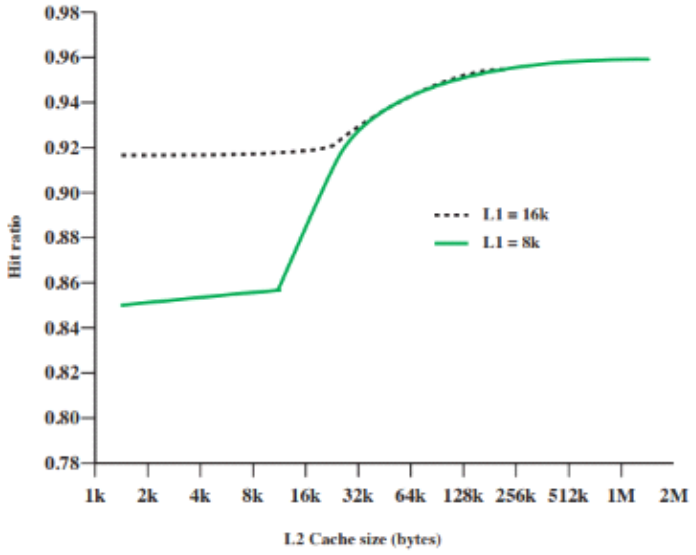
Memory closer to the CPU (Cache - L1, L2, L3) is typically faster but has a far smaller capacity.

Main memory is further and slower but has a far greater capacity

Hard disks have the greatest capacity, and do not need continuous power, but have the slowest access times.

Optimisation: We want to use fastest memory possible for high performance, but problems may large datasets.

Solution: Reuse data already in cache as much as possible and prefetch data from main memory into cache before it is needed, masking load times



Cache Concepts

Caching read-only data is relatively straightforward

We don't need to consider the possibility that items will change, hence copies across the memory hierarchy will remain consistent

1.

Two general strategies can be adopted when caching writes, both of which can employ a write buffer to enhance performance

Write Through - Updates the item in cache and writes through to update lower levels of the memory hierarchy

Write Back - Only updates copy in cache, ensuring that blocks are copied back to memory when when they are to be replaced

Types of Cache Miss

Measures of cache performance include hit rate (h) and miss rate (1-h)

$$h = \frac{\text{Number of times the words are in cache}}{\text{Total number of memory references}}$$

We can understand cache misses by sorting all misses into categories

Compulsory - Misses that would occur regardless of cache size, e.g., the first access to a block can not be in cache (regardless of cache size) so the block must be retrieved

Capacity - Misses occurring because a cache is not large enough to contain all blocks needed during the execution of a program

Conflict - Misses occurring as a result of the placement strategy for blocks not being fully associative, which means that a block may be discarded and retrieved

Coherency - Misses occurring due to cache flushes in multiprocessor systems

Measuring Cache Performance

Performance measures based solely on hit / miss rates don't factor in the cost of a cache miss - the real performance issue!

Average memory access time is an alternative measure that accounts for the cost of a cache miss

$$\text{Average memory access time} = \text{Hit time} + (\text{Miss rate} \times \text{Miss Penalty})$$

This is where hit time is the time to hit in the cache and the miss penalty is the time taken to replace the block from memory

Even average access time is still an indirect measure of performance - execution time can often be a better measure in real-world situations

1. *Larger L1 Cache (16k):*
   - *A larger L1 cache means fewer misses at the L1 level, so the reliance on L2 is reduced.*
   - *Since fewer requests reach the L2 cache, even a small L2 cache is sufficient to achieve a good hit ratio, resulting in a relatively high starting hit ratio.*
2. *Smaller L1 Cache (8k):*
   - *A smaller L1 cache leads to more frequent misses, causing greater reliance on the L2 cache.*
   - *With a small L2 cache, the combined cache system struggles,*

resulting in a lower initial hit ratio.

- *As the L2 cache size grows, it compensates for the smaller L1 cache by reducing overall misses, leading to a sharp improvement in the hit ratio.*

3. Saturation Point (Both Configurations):

- *Beyond a certain L2 cache size (e.g., 128k–256k), the hit ratio flattens out for both configurations.*
- *At this point, the L2 cache is large enough to hold most of the working set of data, so further increases in L2 size do not significantly improve performance.*

Key Takeaways:

- *Larger L1 caches reduce the dependency on L2 caches and provide a higher initial hit ratio.*
- *Smaller L1 caches can still perform well, but they require larger L2 caches to compensate for their higher miss rates.*
- *Beyond a certain L2 cache size, increasing its size has diminishing returns because most misses have already been eliminated.*

1. *In the Write Back caching strategy, data modifications are made only in the cache initially, and the changes are not immediately written to the lower levels of the memory hierarchy (like main memory)*
2. *Instead, the updated data is held in the cache until that specific data block (or cache line) is evicted (replaced by another block because the cache is full).*

1. *When data in a cache block is updated (written to), a dirty bit is set for that block.*
   - *The dirty bit acts as a flag, indicating that the data in the cache block is modified and no longer matches the corresponding data in the lower memory levels.*
2. *The cache keeps the modified data (without writing it back immediately to main memory) to reduce the number of memory write operations.*

- Writing to main memory is slow compared to cache operations, so delaying these writes improves system performance.

3. When the block needs to be replaced (evicted):
   - If the dirty bit is set, the cache writes the updated block back to main memory before replacing it.
   - If the dirty bit is not set (indicating the data wasn't modified), the block can simply be discarded.

1. When a cache miss occurs, it means the data isn't readily available in the cache, and the system has to retrieve it from a lower memory level (like main memory or even further down the memory hierarchy).
2. This retrieval takes significantly more time compared to accessing the cache.
3. The miss penalty is essentially the "extra time" incurred to handle this situation, and it represents the time lost due to the miss because if the data had been in the cache (a hit), this penalty wouldn't exist.

Advantages of Write Back:

- Reduces the number of write operations to the slower main memory, saving time and bandwidth.
- Improves performance in workloads with frequent writes, as the cache absorbs most of the write traffic.

Disadvantages of Write Back:

- Increased complexity due to the need for a dirty bit and mechanisms to handle block replacement.
- Risk of data inconsistency if the cache and main memory are not synchronized properly.

This strategy is especially effective for applications

with a high rate of data reuse, as modified data can stay in the cache and avoid unnecessary writes to main memory.