Memory Hierarchy

In an ideal world, a computer's memory would be instantaneous, infinitely large, and incredibly cheap. In reality, we face a trade-off: fast memory, like Static RAM (SRAM), is extremely expensive and small, while large-capacity storage, like Magnetic Disks, is cheap but orders of magnitude slower. This gap is the central challenge of memory system design. The solution is not a single perfect memory but a Memory Hierarchy.

Principle of Locality. Programs do not access their entire address space randomly. Instead, they tend to focus on a small portion of it at any given time. This behavior is broken into two types:

- *Temporal Locality*: If an item is accessed, it is likely to be accessed again soon (e.g., instructions inside a loop or a variable used repeatedly).
- *Spatial Locality*: If an item is accessed, items with nearby addresses are likely to be accessed soon (e.g., in sequential instruction fetching or when iterating through data in an array).

Hierarchical Structure. The memory hierarchy takes advantage of locality by creating a "pyramid" of memory levels. The levels closer to the processor are smaller, faster, and more expensive, while the levels further away are larger, slower, and cheaper:

(1) *L1 Cache (SRAM)*: The level closest to the CPU. Holds the most recently used data and instructions.

(2) *L2 Cache (DRAM)*: A larger, slightly slower cache that services misses from the L1 cache.

(3) *Main Memory (DRAM)*: The main working memory of the system.

(4) *Storage (Disk/Flash)*: The largest, slowest level that holds everything permanently.

When the CPU needs data, it first checks the L1 cache. If the data is there, it's a Hit. If not, it's a Miss, and the CPU must request the data from the next level down. If the L2 misses, it requests from Main Memory, and so on. The time it takes to fetch data from a lower level and bring it to the upper level is called the Miss Penalty. The goal is to maximize the Hit Ratio (hits/accesses) and minimize the Miss Ratio (misses/accesses).

1.1 Memory Technologies

DRAM. DRAM is the technology used for main memory. Its key characteristics include:

- *Technology*: Stored data as a charge in a tiny capacitor. Because this charge leaks, DRAM must be periodically refreshed (read and written back) to prevent data loss.
- *Organization*: DRAM is organized as a rectangular array of bits. Access involves activating an entire row.
- *Performance*: To improve bandwidth, modern DRAM uses several techniques:

- Synchronous DRAM (SDRAM): Uses a clock to allow for "burst mode", which supplied consecutive words from a row without needing a new address for each one.
- Double Data Rate (DDR): Transfers data on both the rising and falling edges of the clock, effectively doubling the transfer rate.
- DRAM Banking: Allows multiple DRAM chips to be accessed simultaneously to improve bandwidth.

Storage: Flash and Disk. These technologies form the nonvolatile base of the hierarchy.

- *Flash Storage*: A nonvolatile semiconductor memory. It is significantly faster than disk (100x - 1000x) and more robust, but more expensive. NAND flash is the denser, cheaper type used for USB keys and media storage. Its main drawback is that flash bits wear out after thousands of accesses, requiring "wear leveling" techniques.
- *Magnetic Disk*: A nonvolatile, rotating magnetic storage. Accessing a sector of data involves several time-consuming steps: queuing delay, seek time (moving the heads), rotational latency (waiting for the data to rotate under the head), and finally the data transfer.

1.2 Cache Memory

The cache is the part of the memory hierarchy closest to the CPU. When the CPU requests an address, the cache must quickly answer two questions: Is the data already present in the cache? If so, where is it? To manage this, the cache is divided into blocks (also called lines), which are the unit of data transfer. A memory address is subdivided to determine where it fits in the cache:

- *Block Offset*: The low-order bits that identify a specific word or byte within a block.
- *Index*: The middle bits that determine which cache set or line the block maps to.
- *Tag*: The high-order bits that are stored in the cache to identify which specific memory block is currently occupying that line.

A Valid Bit is also stored with each cache line. If this bit is 0, the data in that line is considered not present or invalid.

Cache Mapping. The "where to look" question is answered by the cache's associativity, which defines how flexibly memory blocks can be placed in the cache.

(1) *Direct Mapped*: The simplest and cheapest policy. A memory block can go in only one specific location in the cache, determined by its index. This is very fast to check (only one location), but can cause conflict misses. If the program frequently alternates between two blocks that map to the same index, they will constantly evict each other, leading to misses even if the cache is mostly empty.

(2) *N-Way Set Associative*: A compromise. The cache is divided into sets, each containing `n` blocks (or "ways"). A memory block maps to a single set (based on its index), but it can be placed in any of the `n` locations within that set. This dramatically reduces conflict misses compared to a direct-mapped cache. A 2-way or 4-way associative cache offers most of the benefits. However, it is more complex, as it must search all `n` entries in the set at once.

(3) *Fully Associative*: The most flexible policy. A memory block can be placed in any available location in the entire cache. This eliminates conflict misses entirely, but it is extremely expensive. It requires a comparator for every single entry in the cache, making it practical only for very small caches.

For associative caches, a Replacement Policy is also needed when a set is full. Least-Recently Used (LRU) is a common strategy: the cache evicts the block that has been unused for the longest time.

An Example / Given 64 blocks of cache size and a block size of 16 bytes/block, to what block number does memory address 1200 map?

(1) *Find the Block Address*: First, determine which block the address is in.

$$\text{Block Address} = \left\lfloor \frac{\text{Address}}{\text{Block Size}} \right\rfloor = \left\lfloor \frac{1200}{16} \right\rfloor = 75 \tag{1}$$

(2) *Find the Cache Index (Block Number)*: Next, find where that block maps in the cache using the modulo operator.

$$\text{Block Number} = (\text{Block Address}) \mod (\#\text{ Blocks in Cache}) = 75 \mod 64 = 11 \tag{2}$$

Therefore, address 1200 maps to block 11 of the cache.

Cache Writes. Writing to memory introduces complexity because the cache and main memory can become inconsistent.

On a Write Hit

- *Write-Through*: The data is written to both the cache and main memory at the same time. This is simple but slow, as it makes every write as slow as a memory access. To mitigate this, a write buffer is often used to hold the outgoing data, allowing the CPU to continue immediately without waiting.
- *Write-Back*: The data is written only to the cache block. A dirty bit is set for that block to mark is as modified. The block is only written back to main memory later when it is replaced (and only if it is "dirty"). This is much faster for frequent writes but is more complex to implement.

On a Write Miss

- *Write Allocate*: The block is first fetched from main memory into the cache, and then the write is performed. This is the standard policy for write-back caches.
- *Write Around*: The write bypasses the cache and goes directly to main memory. This is often used with write-through policies, especially for initialization code that writes an entire block without reading it first.

Measuring Cache Performance. The ultimate goal of the memory hierarchy is to improve CPU performance. The time spent stalling for memory is a major component of this.

$$\begin{aligned}\text{CPU Time} &= \text{Program Execution Cycles} + \text{Memory Stall Cycles}\\ \text{Memory Stall Cycles} &= \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss Penalty}\end{aligned} \tag{3}$$

A key metric for evaluating a memory system is the Average Memory Access Time (AMAT):

$$\text{AMAT} = \text{Hit Time} + (\text{Miss Rate} \times \text{Miss Penalty}) \tag{4}$$

As CPUs become faster, the `Miss Penalty` (measured in lost CPU cycles) gets larger. This means that cache performance and a low miss rate become more critical to overall system performance, not less. To bridge the growing gap between fast CPUs and slow DRAM, nearly all modern systems use Multilevel Caches. By splitting the cache, designers can optimize the L1 for speed and the L2 for capacity, achieving better overall performance than a single, large cache.