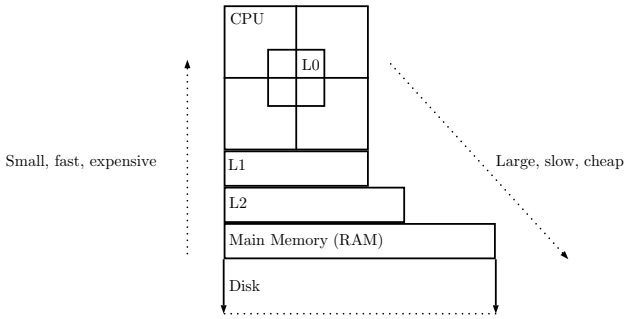
COMPSCI 630 Systems **Spring 2017**

Lecture 6: February 10

Lecturer: Emery Berger *Scribe(s): Sam Baxter, Divyesh Harit*

6.1 Review of Memory Hierarchy



The memory hierarchy is a series of memories, growing in storage capacity and memory access latency as distance from the CPU increases. Modern computer architectures typically have a series of caches built into the CPU (L0, L1, L2, etc.), followed by RAM, and then by disk.

Access to data stored on disk takes orders of magnitudes longer (in CPU cycles) than closer memories, which begs the question of how network latencies are tolerable. Memory access latencies are measured in CPU cycles to retrieve the first byte of data requested. When accessing many bytes sequentially stored, we can *prefetch* the remaining bytes while we wait for the first byte to arrive, hiding the latency so that when the first byte arrives, the rest immediately follow.

6.1.1 Locality

Temporal Locality refers to the reuse of specific data within a small window of time. In other words, data has high temporal locality if when it ccessed, it is likely to be accessed again soon.

Spacial Locality refers to the use of data within relatively close storage locations from each other.

High temporal and/or spacial locality lend themselves to increased performance due to caches by hiding the latency of memory accesses by keeping things often or likely to be used in low-latency caches. Memory that exhibits low locality leads to cache misses on memory accesses, incurring overhead and degredating performance of caches.

6.2 Cache Replacement Policies

Cache performance is determined by the rate of cache hits and misses. Common reasons for misses are:

- **Compulsory Misses** - The data has not been accessed previously
- **Capacity Misses** - The data previously resided in cache, but has been evicted due to space capacity.

Cache misses degrade performance and miss rates are subject to how we evict data from caches, known as the **Cache Replacement Policy**.

6.2.1 LRU - Least Recently Used

LRU caches evict the oldest cache entry when capacity has been reached and a cache miss occurs.

6.2.2 MRU - Most Recently Used

MRU caches evict the newest cache entry. This seems counterintuitive, and is rarely used in systems.

6.2.3 RAND - Random

RAND caches nondeterministically evict a random entry on cache misses. It can be shown that RAND and LRU converge in the worst-case. RAND is actually desirable in some scenarios, such as when many entries are reused often, and a small number have recently but infrequently been used.

6.2.4 OPT - Optimal Behavior

OPT needs to know the future, but provides the optimal cache hit rate for a given trace of memory accesses. The process is to:

- Run the program once to collect a trace of accesses.
- Traverse the trace in reverse and evict the entry which has not been used in the longest amount of time.

OPT is impractical for general use, but is useful as the ideal baseline for comparing all other cache replacement algorithms.

6.2.5 Implementation

Chips typically implement an approximation of LRU. Strict LRU is not used because this requires a linear search in the size of the cache to find the least recently used object at each point of eviction.

Instead, chips divide the cache into *sets* according to hashes on the address of objects. Each *set* is divided into some number of *ways*, which are treated as ages or generations using a small number of bits to approximate age. To choose an entry to evict, one computes the hash of the new entry to determine the set, and evicts any entry in the oldest age (i.e. way).

Performance depends on the quality of hash function. Evictions that are caused because two entries map to the same hash are called **conflict misses**.

One-way (Direct Mapped) caches have a single *way* per set, and the number of sets is equal to the capacity of the cache.

Fully Associative caches have a single set, and the number of *ways* is equal to the capacity of the cache. This behavior corresponds to LRU.

6.2.6 TLB

The TLB is a unique fully associative cache on CPUs. It stores the mapping of virtual to physical addresses in what is called a **page table**. Pages are typically 4K chunks rather than cache line sized. Because paging must be high performance and the TLB implements strict LRU, the TLB size is typically small, limited to 256 or 512 entries.

6.3 Instruction Set Architectures (ISAs)

6.3.1 Reduced vs. Conventional

The 1970s and 1980s saw a growth in programming language and application-specific architectures, including a large number of high-level abstractions as machine instructions.

The simplified FETCH-DECODE-EXECUTE processor cycle can incur latency in the DECODE portion due to complex instructions. The **RISC** (Reduced instruction set computer) architecture offered a solution to this problem.

RISCs are composed of a core set of instructions of the same size and execution time. This greatly simplified the CPU cycle.

The modern x86 architecture is a hybrid CISC/RISC, compiling complex instructions to **micro ops** on the chip. Fetches and decodes happen once and are cached.