ECE254: Operating Systems and Systems Programming Spring 2015

Lecture 22 — Caching

Jeff Zarnett

Caching

Caching is very... hit and miss.

Caching is very important in computing, and not just memory. We examine the idea of caching in the context of memory, but it is applicable any time there is a large resource that is divided into pieces, some of which are used more often than others. The goal of caching is to speed up operations. It is desirable to read information from cache, when possible, because it takes less time to get data from cache to the CPU than from main memory to the CPU. CPUs are a lot faster than memory and it is best if we do not keep them waiting.

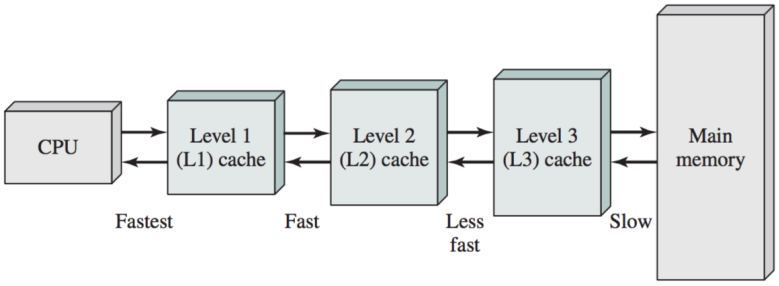
Caches do not have to operate on pages; they can operate on anything, but they are typically blocks of a given size. An entry in a cache is often called a *line*. We will assume for the balance of this discussion that a cache line maps nicely to a page.

As discussed, the CPU generates a memory address for a read or write operation. The address will be mapped to a page. Ideally, the page is found in the cache, because that would be faster. If the requested page is, in fact, in the cache, we call that a cache *hit*. If the page is not found in the cache, it is considered a cache *miss*. In case of a miss, we must load the page from memory, a comparatively slow operation. The percentage of the time that a page is found in the cache is called the *hit ratio*, because it is how often we have a cache hit. We can calculate the effective access time if we have a good estimate of the hit ratio (which is not overly difficult to obtain) and some measurements of how long it takes to load data from the cache and how long from memory. The effective access time is therefore computed as:

$$\text{Effective Access Time} = h \times t_c + (1 - h) \times t_m$$

Where h is the hit ratio, t_c is the time required to load a page from cache, and t_m is the time to load a page from memory. Of course, we would like the hit ratio to be as high as possible.

Caches have limited size, because faster caches are more expensive. With infinite money we might put everything in registers, but that is rather unrealistic. Caches for memory are very often multileveled; Intel 64-bit CPUs tend to have L1, L2, and L3 caches. L1 is the smallest and L3 is the largest. See the diagram below:



Three levels of cache between the CPU and main memory [Sta14].

If we have a miss in the L1 cache, the L2 cache is checked. If the L2 cache contains the desired page, it will be copied to the L1 cache and sent to the CPU. If it is not in L2, then L3 is checked. If it is not there either, it is in main memory and will be retrieved from there and copied to the in-between levels on its way to the CPU. Because caches have limited size, we have to manage this carefully.

Page Replacement Algorithms

Whenever a page fault occurs, the operating system needs to choose which page to evict from the cache to make space for the new one.

References

[Sta14] William Stallings. *Operating Systems Internals and Design Principles (8th Edition)*. Prentice Hall, 2014.