УДК 004.254
Ilya Sheverdinkin[1], Olha Kalantaieva[2]
[1]student of group CST-118 ZNTU
[2]teacher ZNTU

CACHE MEMORY

In computing, a cache is a hardware or software component that stores data so that future requests for that data can be served faster; the data stored in a cache might be the result of an earlier computation or a copy of data stored elsewhere. A cache hit occurs when the requested data can be found in a cache, while a cache miss occurs when it cannot. Cache hits are served by reading data from the cache, which is faster than recomputing a result or reading from a slower data store; thus, the more requests that can be served from the cache, the faster the system performs.

To be cost-effective and to enable efficient use of data, caches must be relatively small. Nevertheless, caches have proven themselves in many areas of computing, because typical computer applications access data with a high degree of locality of reference. Such access patterns exhibit temporal locality, where data is requested that has been recently requested already, and spatial locality, where data is requested that is stored physically close to data that has already been requested.

A CPU cache is a hardware cache used by the central processing unit (CPU) of a computer to reduce the average cost (time or energy) to access data from the main memory. A cache is a smaller, faster memory, closer to a processor core, which stores copies of the data from frequently used main memory locations. Most CPUs have different independent caches, including instructions and data caches, where the data cache is usually organized as a hierarchy of more cache levels (L1, L2, L3, L4, etc.).

All modern CPUs have multiple levels of CPU caches. The first CPUs that used a cache had only one level of cache; unlike later level 1 caches, it was not split into L1d (for data) and L1i (for instructions). Almost all current CPUs with caches have a split L1 cache. They also have L2 caches and, for larger processors, L3 caches as well. The L2 cache is usually not split and acts as a common repository for the already split L1 cache. Every core of a multi-core processor has a dedicated L2 cache and is usually not shared between the cores. The L3 cache, and higher-level caches, are shared between the cores and are not split. An L4 cache is currently uncommon, and is generally on dynamic random-access memory (DRAM), rather than on static random-access memory (SRAM), on a separate die or chip. That was also the case historically with L1, while bigger chips have allowed integration of it and generally all cache levels, with the possible exception of the last level. Each extra level of cache tends to be bigger and be optimized differently.

Other types of caches exist (that are not counted towards the "cache size" of the most important caches mentioned above), such as the translation look aside buffer (TLB) that is part of the memory management unit (MMU) that most CPUs have.

Caches are generally sized in powers of two: 4, 8, 16 etc. KiB or MiB (for larger non-L1) sizes, although the IBM z13 has a 96 KiB L1 instruction cache.

Earlier graphics processing units (GPUs) often had limited read-only texture caches, and introduced morton order swizzled textures to improve 2D cache coherency. Cache misses would drastically affect performance, e.g. if mipmapping was not used. Caching was important to leverage 32-bit (and wider) transfers for texture data that was often as little as 4 bits per pixel, indexed in complex patterns by arbitrary UV coordinates and perspective transformations in inverse texture mapping.

As GPUs advanced they have developed progressively larger and increasingly general caches, including instruction caches for shaders, exhibiting increasingly common functionality with CPU caches. For example, GT200 architecture GPUs did not feature an L2 cache, while the Fermi GPU has 768 KB of last-level cache, the Kepler GPU has 1536 KB of last-level cache, and the Maxwell GPU has 2048 KB of last-level cache. These caches have grown to handle synchronization primitives between threads and atomic operations, and interface with a CPU-style MMU.