TEMPLATE FOR DEFINITIONAL ENTRY

Processor Cache

Peter Boncz
Centrum voor Wiskunde en Informatica (CWI)
Kruislaad 413, 1098 SJ Amsterdam, The Netherlands
boncz@cwi.nl

SYNONYMS

Data cache, Instruction cache, CPU cache, L1 cache, L2 cache, L3 cache, Translation Lookaside Buffer (TLB)

DEFINITION

To hide the high latencies of DRAM access, modern computer architecture now features a memory hierarchy that besides DRAM also includes SRAM cache memories, typically located on the CPU chip. Memory access first check these caches, which takes only a few cycles. Only if the needed data is not found, an expensive memory access is needed.

MAIN TEXT

CPU caches are SRAM memories located on the CPU chip, intended to hide the high latency of accessing off-chip DRAM memory. Caches are organized in cache lines (typically 64 bytes). In a fully-associative cache, each memory line can be stored in any location of the cache. To make checking the cache fast, however, CPU caches tend to have limited associativity, such that storage of a particuar cache line is possible in only 2 or 4 locations. Thus only 2 or 4 locations need to be checked during lookup (these are called 2- resp. 4-way associative caches). The cache hit ratio is determined by the spatial and temporal locality of the memory accesses generated by the running program(s).
Cache misses can either be compulsory misses (getting the cache lines of all used memory once), capacity misses (caused by the cache being too small to keep all multiply used lines in cache), or conclict misses (due to the limited associativity of the cache).

Most modern CPUs have at least three independent caches: an instruction cache to speed up executable instruction fetch, a data cache to speed up data fetch and store, and a Translation Lookaside Buffer (TLB) used to speed up virtual-to-physical address translation for both executable instructions and data. The TLB is not organized in cache lines, it simply holds pairs of (virtual,logical) page mappings, typically a fairly limited amount (e.g. 64). In practice, this mean that algorithms that repeatedly touch memory in more than 64 pages (whose size is often 4KB) shortly after each other, run into TLB thrashing. This problem can sometimes be mitigated by setting a large virtual memory page size, or by using special large OS pages (sometimes supported in the CPU with a separate, smaller, TLB for large pages).

Another issue is the tradeoff between latency and hit rate. Larger caches have better hit rates but longer latency. To address this tradeoff, many computers use multiple levels of cache, with small fast caches backed up by larger slower caches. Multi-level caches generally operate by checking the smallest Level 1 (L1) cache first; if it hits, the processor proceeds at high speed. If the smaller cache misses, the next larger cache (L2) is checked, and so on, before external memory is checked. As the latency difference between main memory and the fastest cache has become larger, some processors have begun to utilize as many as three levels of on-chip cache.

For multi-CPU and multi-core systems, the fact that some of the higher levels of cache are not shared, yet provide coherent access to shared memory , causes additional cache-coherency inter-core communication to invalid stale copies of cache lines on other cores when one core modifies it. In multi-core CPUs, an important issue is which cache level is shared among all cores – this cache level is on the one hand a potential hot-spot for cache conflicts, on the other hand provides an opportunity for very fast inter-core data exchange.

In case of sequential data processing, the memory controller or memory chipset in modern computers often detect this access pattern and start requesting the subsequent cache lines in advance. This is called hardware prefetching. Prefetching effectively allows to hide compulsory cache misses. Without prefetching, the effective memory bandwidth would equate cache line size divided my memory latency (e.g. 64/50ns = 1.2GB/s). Thanks to hardware prefetching, modern computer architectures reach four times that on sequential access. Modern CPUs also offer explicit prefetching instructions, which a software writer can exploit to perform (non-sequential) memory accesses in advance, hiding their latency. In database systems, such software prefetching has successfully been used in making hash-table lookup faster (e.g. in hash-join and hash-aggregation).

In database systems, a series of cache-conscious data storage layouts (e.g. DSM and PAX) have been proposed to improve cache line usage. Also, a number of cache-conscious query processing algorithms, such as cache-partitioned hash join and hash-join using memory prefetching, have been studied. In the area of data structures and theoretical computer science, there has recently been interest in cache-oblivious algorithms, that regardless the exact parameters of the memory hierarchy (number of levels, cache size, cache line sizes and latencies) perform well.

CROSS REFERENCES

Main Memory
Disk
Main Memory Databases
Architecture-conscious databases
Cache-conscious algorithms
DSM, PAX storage layouts