CACHE HIERARCHY IN MODERN PROCESSORS AND ITS IMPACT ON COMPUTING

Subrahmanya Bhat B *.
Dr. K.R Kamath**.
*Dept MCA, Srinivas Institute of Mgt Studies, Pandeshwara, Mangaluru,, Karnataka- 575001, Email: itsbhat@gmail.com
**Dept CS, Srinivas Institute of Technology, Mangaluru, Karnataka.

ABSTRACT

Modern processors have multiple interacting caches on chip. Pipelined CPUs access memory from multiple points in the Pipeline - Instruction Fetch, Virtuval to Physical address translation, and Data fetch. The natural design is to use different physical caches for each of these points. Thus the pipeline naturally ends up with at least three separate caches (instruction, TLB, and data). This implementation needs machines with special architecture like separate instruction and data memories. Most modern CPUs have a single-memory von Neumann Architecture. A victim cache is a cache used to hold blocks evicted from a CPU cache upon replacement. The victim cache lies between the main cache and its refill path, and only holds blocks that were evicted from the main cache. One of the more extreme examples of cache specialization is the trace cache found in the Intel Pentium 4 microprocessors. A trace cache is a mechanism for increasing the instruction fetch bandwidth and decreasing power consumption by storing traces of instructions that have already been fetched and decoded. 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. Some processors, all data in the L1 cache must also be somewhere in the L2 cache. These caches are called strictly inclusive. Other processors (like the AMD Athlon) have exclusive caches — data is guaranteed to be in at most one of the L1 and L2 caches, never in both. The advantage of exclusive caches is that they store more data. This advantage is larger when the exclusive L1 cache is comparable to the L2 cache, and diminishes if the L2 cache is many times larger than the L1 cache. One advantage of strictly inclusive caches is that when external devices or other processors in a multiprocessor system wish to remove a cache line from the processor, they need only have the processor check the L2 cache. In cache hierarchies which do not enforce inclusion, the L1 cache must also be checked.

Keywords: Cache, Victim Cache, Trace Cache, Inclusive Cache, Exclusive Cache

Introduction:

The cache is a smaller, faster memory which stores copies of the data from the most frequently used main memory locations. Cache memory is used by the CPU of a computers to reduce the average time to access memory. As long as most memory accesses are cached memory locations, the average latency of memory accesses will be closer to the cache latency than that of main memory. When the processor needs to read from or write to a location in main memory, it first checks whether a copy of that data is in the cache. If so, the processor immediately reads from or writes to the cache, which is much faster than reading from or writing to main memory.

Most modern desktop and server 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.

Each location in each memory contains data (a *cache line*), which in different designs may range in size from 8 to 512 bytes. The size of the cache line is usually larger than the size of the usual access requested by a CPU instruction, which ranges from 1 to 16 bytes. The largest addresses and data handled by current 32 bits and 64 bits architectures being 128 bits long i.e. 16 bytes. Each location in each memory also has an index, which is a unique number used to refer to that location. The index for a location in main memory is called an address. Each location in the cache has a tag that contains the index of the datum in main memory that has been cached. In a CPU's data cache these entries are called *cache lines* or *cache blocks*.

When the processors needs to read or write a location in main memory, it first checks whether that memory location is in the cache. This is accomplished by comparing the address of the memory location to all tags in the cache that might contain that address. If the processor finds that the memory location is in the cache, then it is a *cache hit;* otherwise, *cache miss*. In the case of a cache hit, the processor immediately reads or writes the data in the cache line. The proportion of accesses that result in a cache hit is known as the *hit rate*, and is a measure of the effectiveness of the cache for a given program or algorithm.

In the case of a miss, the cache allocates a new entry, which comprises the tag just missed and a copy of the data. The reference can then be applied to the new entry just as in the case of a hit. Read misses delay execution because they require data to be transferred from a much slower memory than the cache itself. Write misses may occur without such penalty since the data can be copied in the background. Instruction caches are similar to data caches but the CPU only performs read accesses (instruction fetch) to the instruction cache. Instruction and data caches can be separated for higher performance.

In order to make room for the new entry on a cache miss, the cache has to *evict* one of the existing entries. One popular replacement policy is LRU which replaces the least recently used entry. If data are written to the cache, they must at some point be written to main memory as well. The timing of this write is controlled by *write policy*. In a *write-through* cache, every write to the cache causes a write to main memory. Alternatively, in a *write-back* or *copy-back* cache, writes are not immediately mirrored to the main memory. Instead, the cache tracks which locations have been written over (these locations are marked *dirty*). Alternatively, when the CPU in a multi-core processor updates the data in the cache, copies of data in caches associated with other cores will become stale. Communication protocols between the cache managers which keep the data consistent are known as Cache Coherence Protocols.

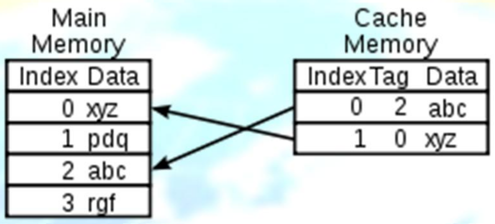

Fig 1. Memory to Cache Mapping

Cache Design in Modern Processors:

Most of the modern processors have multiple interacting caches on chip like **Specialized caches, Victim cache, Trace cache, Multi-level caches, Exclusive versus inclusive**

Specialized caches

Pipelined CPUs access memory from multiple points in the pipeline like, instruction fetch, virtuval to physical address translation, and data fetch. The natural design is to use different physical caches for each of these points, so that no one physical resource has to be scheduled to service two points in the pipeline. Thus the pipeline naturally ends up with at least three separate caches (instruction, TLB and data), each specialized to its particular role. Pipelines with separate instruction and data caches, now predominant, are said to have a Harvard Architecture.

Victim cache:

A **victim cache** is a cache used to hold blocks evicted from a CPU cache upon replacement. The victim cache lies between the main cache and its refill path, and only holds blocks that were evicted from the main cache. The victim cache is usually fully associative, and is intended to reduce the number of conflict misses. Many commonly used programs do not require an associative mapping for all the accesses. In fact, only a small fraction of the memory accesses of the program require high associativity. The victim cache exploits this property by providing high associativity to only these accesses and improves the system performance.

Trace cache:

One of the more extreme examples of cache specialization is the **trace cache** found in the Intel Pentium 4 microprocessors. A **trace cache** is a mechanism for increasing the instruction fetch bandwidth and decreasing power consumption (in the case of the Pentium 4) by storing traces of instructions that have already been fetched and decoded.

A trace cache stores instructions either after they have been decoded, or as they are retired. Generally, instructions are added to trace caches in groups representing either individual basic blocks or dynamic instruction traces. A dynamic trace ("trace path") contains only instructions whose results are actually used, and eliminates instructions following taken branches (since they are not executed); a dynamic trace can be a concatenation of multiple basic blocks. This allows the instruction fetch unit of a processor to fetch several basic blocks, without having to worry about branches in the execution flow.

Trace lines are stored in the trace cache based on the program counter of the first instruction in the trace and a set of branch predictions. This allows for storing different trace paths that start on the same address, each representing different branch outcomes. In the instruction fetch stage of a pipeline, the current program counter along with a set of branch predictions is checked in the trace cache for a hit. If there is a hit, a trace line is supplied to fetch which does not have to go to a regular cache or to memory for these instructions. If there is a miss, a new trace starts to be built.

Trace caches are also used in Pentium processors to store already decoded micro-operations, or translations of complex x86 instructions, so that the next time an instruction is needed, it does not have to be decoded again and hence contribute to the system performance.

Multi-level caches:

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. Finally, at the other end of the memory hierarchy, the CPU register file itself can be considered the smallest, fastest cache in the system, with the special characteristic that it is scheduled in software, typically by a compiler, as it allocates registers to hold values retrieved from main memory.

Exclusive versus inclusive:

Multi-level caches introduce a new design decisions. For instance, in some processors, all data in the L1 cache must also be somewhere in the L2 cache. These caches are called **Strictly inclusive**. Other processors (like the AMD Athlon) have **Exclusive caches** where data is guaranteed to be in at most one of the L1 and L2 caches, but not in both.

The advantage of exclusive caches is that they store more data. This advantage is larger when the exclusive L1 cache is comparable to the L2 cache, and diminishes if the L2 cache is many times larger than the L1 cache. When the L1 misses and the L2 hits on an access, the hitting cache line in the L2 is exchanged with a line in the L1. This exchange is quite a bit more work than just copying a line from L2 to L1, which is what an inclusive cache does. One advantage of strictly inclusive caches is that when external devices or other processors in a multiprocessor system wish to remove a cache line from the processor, they need only have the processor check the L2 cache. In cache hierarchies which do not enforce inclusion, the L1 cache must be checked as well and it affects the system performance.

To illustrate both specialization and multi-level caching, here is the cache hierarchy of the K8 core in the AMD Athlon64 CPU.

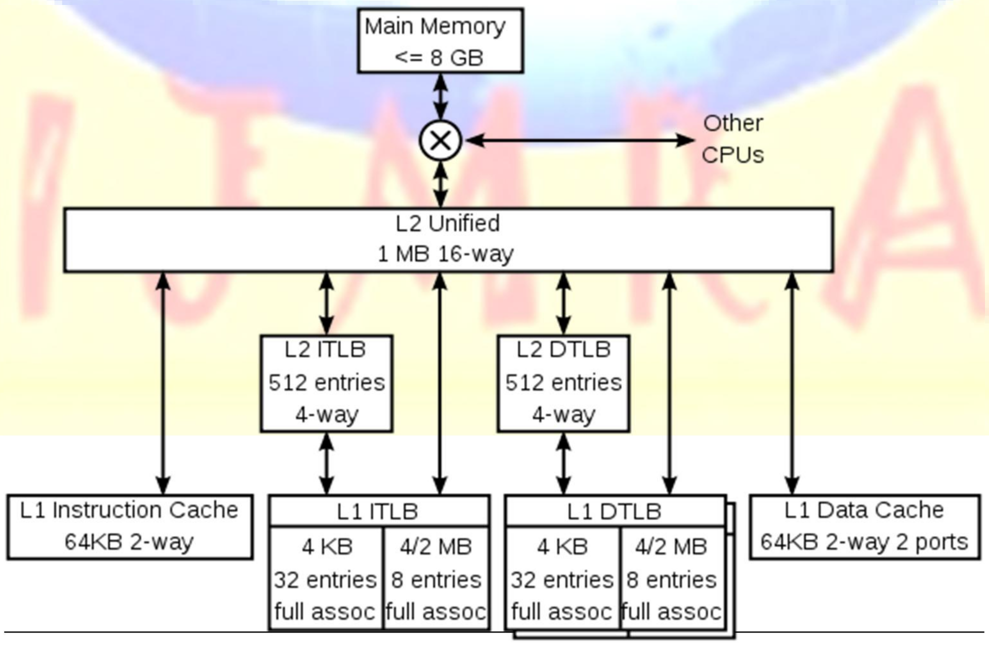

Fig 2: AMD Athlon64 CPU

Conclusion:

In reducing the Memory latency Cache memory has been designed in CPU architecture. But the recent developments in both hardware and software haslead to a new design appraches to the Cache. Introducing Specialized Cache has improved the system performance especially in Pipe Line Architecture with at least three separate caches Instruction Cache, TLB Cache and Data Cache. A **victim cache** is a cache used to hold blocks evicted from a CPU cache upon replacement. The victim cache is usually fully associative, and is intended to reduce the number of conflict misses. A **trace cache** is a mechanism for increasing the instruction fetch bandwidth and decreasing power consumption by storing traces of instructions that have already been fetched and decoded. The advantage of **Multi Level - Exclusivecache** is that they store more data. This advantage is larger when the exclusive L1 cache is comparable to the L2 cache. The advantage of **Multi Level - Inclusivecache** is that when external devices or other processors in a multiprocessor system wish to remove a cache line from other processor, they need only have to check the L2 cache and not both.

References:

1. André Seznec. "A Case for Two-Way Skewed-Associative Caches", http://dx.doi.org/10.1145/173682.165152. Retrieved 2007-12-13.
2. "Advanced Caching Techniques" by C. Kozyrakis
3. Micro-Architecture "Skewed-associative caches have ... major advantages over conventional set-associative caches."
4. Cache performance of SPEC CPU2000"".Cs.wisc.edu. http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data/. Retrieved 2010-05-02.
5. Linux Journal. http://www.linuxjournal.com/article/7105. Retrieved 2010-05-02. http://www.systems.ethz.ch/education/courses/fs09/aos/lectures/wk3-print.pdf
6. "AMD K8". Sandpile.org. http://www.sandpile.org/impl/k8.htm.Retrieved 2007-06-02.
7. "The Processor-Memory performance gap". acm.org. http://www.acm.org/crossroads/xrds5-3/pmgap.html.Retrieved 2007-11-08.
8. "Chip Design Thwarts Sneak Attack on Data" by Sally Adee 2009 discusses "A novel cache architecture with enhanced performance and security" [1][2] by Zhenghong Wang and Ruby B. Lee: (abstract) "Caches ideally should have low miss rates and short access times, and should be power efficient at the same time. Such design goals are often contradictory in practice."