Memory Ordering in Modern Microprocessors, Part I

Jun 30, 2005 By Paul E. McKenney (/user/1001349)

in

One important difference among CPU families is how they allow memory accesses to be reordered. Linux has to support them all.

(/issue/136)

From Issue #136 August 2005 (/issue/136)

Since the 2.0 kernel release, Linux has supported a large number of SMP systems based on a variety of CPUs. Linux has done an excellent job of abstracting differences among these CPUs, even in kernel code. This article is an overview of one important difference: how CPUs allow memory accesses to be reordered in SMP systems.

Memory accesses are among the slowest of a CPU's operations, due to the fact that Moore's law has increased CPU instruction performance at a much greater rate than it has increased memory performance. This difference in performance increase means that memory operations have been getting increasingly expensive compared to simple register-to-register instructions. Modern CPUs sport increasingly large caches in order to reduce the overhead of these expensive memory accesses.

These caches can be thought of as simple hardware hash tables with fixed size buckets and no chaining, as shown in Figure 1. This cache has 16 lines and two ways for a total of 32 entries, each entry containing a single 256-byte cache line, which is a 256-byte-aligned block of memory. This cache line size is a little on the large size, but it makes the hexadecimal arithmetic much simpler. In hardware parlance, this is a two-way set-associative cache. It is analogous to a software hash table with 16 buckets, where each bucket's hash chain is limited to two elements at most. Because this cache is implemented in hardware, the hash function is extremely simple: extract four bits from the memory address.

In Figure 1, each box corresponds to a cache entry that can contain a 256-byte cache line. However, a cache entry can be empty, as indicated by the empty boxes in the figure. The rest of the boxes are flagged with the memory address of the cache line they contain. Because the cache lines must be 256-byte aligned, the low eight bits of each address are zero. The choice of hardware hash function means the next-higher four bits match the line number.

The situation depicted in Figure 1 might arise if the program's code was located at address 0x43210E00 through 0x43210EFF, and this program accessed data sequentially from 0x12345000 through 0x12345EFF. Suppose that the program now was to access location 0x12345F00. This location hashes to line 0xF, and both ways of this line are empty, so the corresponding 256-byte line can be accommodated. If the program was to access location 0x1233000, which hashes to line 0x0, the corresponding 256-byte cache line can be accommodated in way 1. However, if the program were to access location 0x1233E00, which hashes to line 0xE, one of the existing lines must be ejected from the cache to make room for the new cache line. This background on hardware caching allows us to look at why CPUs reorder memory accesses.

	Way 0	Way 1
0x0	0x12345000	
0x1	0x12345100	
0x2	0x12345200	
0x3	0x12345300	
0x4	0x12345400	
0x5	0x12345500	
0x6	0x12345600	
0x7	0x12345700	
0x8	0x12345800	
0x9	0x12345900	
0xA	0x12345A00	
0xB	0x12345B00	
0xC	0x12345C00	
0xD	0x12345D00	
0xE	0x12345E00	0x43210E00
0xF		

(/files/linuxjournal.com/linuxjournal/articles/082/8211/8211f1.jpg)

Figure 1. CPU Cache Structure for a Cache with 16 Lines and Two Entries Per Line

Why Reorder Memory Accesses?

In a word, performance! CPUs have become so fast that the large multimegabyte caches cannot keep up with them. Therefore, caches often are partitioned into nearly independent banks, as shown in Figure 2. This allows each of the banks to run in parallel, thus keeping up better with the CPU. Memory normally is divided among the cache banks by address. For example, all the even-numbered cache lines might be processed by bank 0 and all of the odd-numbered cache lines by bank 1.

However, this hardware parallelism has a dark side: memory operations now can complete out of order, which can result in some confusion, as illustrated in Figure 3. CPU 0 might write first to location 0x12345000, an even-numbered cache line, and then to location 0x12345100, an odd-numbered cache line. If bank 0 is busy with earlier requests but bank 1 is idle, the first write is visible to CPU 1 after the second write. In other words, the writes are perceived out of order by CPU 1. Reads can be reordered in a similar manner. This reordering can cause many textbook parallel algorithms to fail.

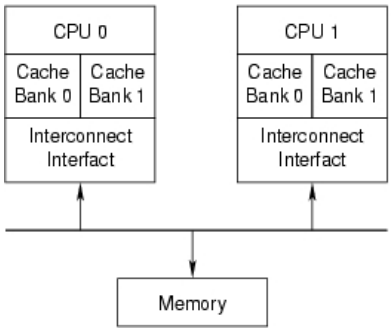


Figure 2. Hardware parallelism divides one large cache into multiple banks.

Memory Reordering and SMP Software

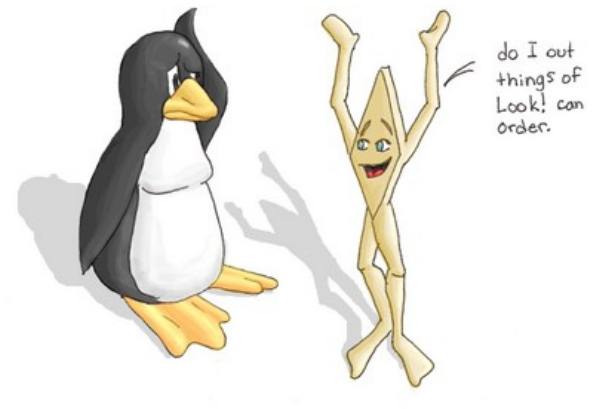
A few machines offer sequential consistency, in which all operations happen in the order specified by the code and where all CPUs' views of these operations are consistent with a global ordering of the combined operations. Sequentially consistent systems have some nice properties, but high performance does not tend to be one of them. The need for global ordering severely constrains the hardware's ability to exploit parallelism, and therefore, commodity CPUs and systems do not offer sequential consistency.

On these systems, three orderings must be accounted for:

1. Program order: the order in which the memory operations are specified in the code running on a given CPU.

2. Execution order: the order in which the individual memory-reference instructions are executed on a given CPU. The execution order can differ from program order due to both compiler and CPU-implementation optimizations.

3. Perceived order: the order in which a given CPU perceives its and other CPUs' memory operations. The perceived order can differ from the execution order due to caching, interconnect and memory-system optimizations. Different CPUs might well perceive the same memory operations as occurring in different orders.



(/files/linuxjournal.com/linuxjournal/articles/082/8211/8211f3.resized.jpg)

Figure 3. CPUs can do things out of order.

Popular memory-consistency models include x86's process consistency, in which writes from a given CPU are seen in order by all CPUs, and weak consistency, which permits arbitrary reorderings limited only by explicit memory-barrier instructions. For more information on memory-consistency models, see Gharachorloo's exhaustive technical report, listed in the on-line Resources.

Comments

Comment viewing options

Threaded list - expanded ▼ | Date - newest first ▼ | 50 comments per page ▼ | Save settings

Select your preferred way to display the comments and click "Save settings" to activate your changes.

First article I've seen on (/article/8211#comment-355160)

Submitted by RossC (not verified) on Sun, 08/22/2010 - 16:31.

First article I've seen on CPU reordering that explained *why* it happens. Great stuff.