registers are unlike main memory, because memory just holds data and cannot modify it or take actions based on it.

Whereas caching an I/O region can cause system disruption, caching certain legitimate main memory regions can cause performance degradation due to thrashing. It may not be worth caching small routines that are infrequently executed, because the performance benefit of caching a quick maintenance routine may be small, and its effect on flushing more valuable cache entries may significantly slow down the application that resumes execution when the maintenance routine completes. A performance-critical application is often composed of a processing kernel along with miscellaneous initialization and maintenance routines. Most of the microprocessor time is spent executing kernel instructions, but sometimes the kernel must branch to maintenance routines for purposes such as loading or storing data. Unlike I/O regions that are inherently known to be cache averse, memory regions that should not be cached can only be known by the programmer and explicitly kept out of the cache.

Methods of excluding certain memory locations from the cache differ across system implementations. A cache controller will often contain a set of registers that enable the lockout of specific memory regions. On those integrated microprocessors that contain some address decoding logic as well as a cache controller, individual memory areas are configured into the decoding logic with programmable registers, and each is marked as cacheable or noncacheable. When the microprocessor performs a memory transaction, the address decoder sends a flag to the cache controller that tells it whether to participate in the transaction.

On the flip side of locking certain memory regions out of the cache, some applications can benefit from explicitly locking certain memory regions into the cache. Locking cache entries prevents the cache controller from flushing those entries when a miss occurs. A programmer may be able to lock a portion of the processing kernel into the cache to prevent arbitrary maintenance routines from disturbing the most frequently accessed sets of instructions and data.

Cache controllers perform burst transactions to main memory because of their multiword line architecture. Whether the cache is reading a new memory block on a cache miss or writing a dirty block back to main memory, its throughput is greatly increased by performing burst transfers rather than reading or writing a single word at a time. Normal memory transfers are executed by presenting an address and reading or writing a single unit of data. Each type of memory technology has its own associated latency between the address and data phases of a transaction. SRAM is characterized by very low access latency, whereas DRAM has a higher latency. Because main memory in most systems is composed of DRAM, single-unit memory transfers are inefficient, because each byte or word is penalized by the address phase overhead. Burst transfers, however, return multiple sequential data units while requiring only an initial address presentation, because the address specifies a starting address for a set of memory locations. Therefore, the overhead of the address phase is amortized across many data units, greatly increasing the efficiency of the memory system. Modern DRAM devices support burst transfers that nicely complement the cache controllers that often coexist in the same computer system.

As a result of cache subsystems being integrated onto the same chip along with high-performance microprocessors, the external memory interface is less a microprocessor bus and more a burst-mode cache bus. The microprocessor needs to be able to bypass the cache controller while accessing noncacheable memory locations or during boot-up when peripherals such as the cache controller have not yet been initialized. However, the external bus is often optimized for burst transfers, and absolute efficiency or simplicity when dealing with noncacheable locations may be a secondary concern to the manufacturer. If overall complexity can be reduced by giving up some performance in nonburst transfers, it may be worth the trade-off, because high performance microprocessors spend relatively little of their time accessing external memory directly. If they do, then something is wrong with the system design, because the microprocessor's throughput is sure to suffer.

Many microprocessors are designed to support multiple levels of caching to further improve performance. In this context, the cache that is closest to the microprocessor core is termed a *level-one*

(L1) cache. The L1 cache is fairly small, anywhere from 2 kB to 64 kB, with its benefit being speed. Because it is small and close to the microprocessor, it can be made to run as fast as the microprocessor can fetch instructions and data. Instruction and data caches are implemented at L1. Line sizes for L1 caches vary but are often 16 or 32 bytes. The line size needs to be kept to a practical minimum to maximize the number of unique memory blocks that can be stored in a small RAM structure.

Level two (L2) caches may reside on the same silicon chip as the microprocessor and L1 cache or externally to the chip, depending on the implementation. L2 caches are generally unified instruction and data caches to minimize the complexity of the interface between the L1 cache and the rest of the system. These caches run somewhat slower than L1 but, consequently, they can be made larger: 128 kB, 256 kB, or more. Line sizes of 64 bytes and greater are common in L2 caches to increase efficiency of main memory burst transfers. Because the L2 cache has more RAM, it can expand both the number of lines and the line size in the hope that, when the L1 cache requests a block of memory, the next sequential block will soon be requested as well. Beyond L2, some microprocessors support L3 and even L4 caches. Each successive level increases its latency of response to the cache above it but adds more cache RAM (sometimes megabytes) and sometimes larger line sizes to increase burst-mode transfer efficiency to main memory.

As core microprocessor clock frequencies commonly top several hundred megahertz, and the most advanced microprocessors exceed 1 GHz, the bandwidth disparity between the microprocessor and main memory increases. Cache misses impose severe penalties on throughput, because the effective clock speed of the microprocessor is essentially reduced to that of the memory subsystem when data is fetched directly from memory. The goal of a multilevel cache structure is to substantially reduce the probability of a cache miss that leads directly to main memory. If the L1 cache misses, hopefully, the L2 cache will be ready to supply the requested data at only a moderate throughput penalty.

Caching as a concept is not restricted to the context of microprocessors and hardware implementation. Caches are found in hard disk drives and in Internet caching products. Some high-end hard drives implement several megabytes of RAM to prefetch data beyond that which has already been requested. While the hard drive's cache, possibly implemented using DRAM, is not nearly as fast as a typical microprocessor, it is orders of magnitude faster than the drive mechanism itself. Internet caching products routinely copy commonly accessed web sites and other data onto their hard drives so that subsequent accesses do not have to go all the way out to the remote file server. Instead, the requested data is sitting locally on a cache system. Caching is the general concept of substituting a small local storage resource that is faster than the larger more remote resource. Caches can be applied in a wide variety of situations.

Caching gets somewhat more complex when the data that is being cached can be modified by another entity outside of the cache memory. This is possible in the Internet caching application mentioned above or in a multiprocessor computer. Cache coherency is the subject of many research papers and is a problem that needs to be addressed by each implementation. Simply put, how does the Internet cache know when a web site that is currently stored has been updated? A news web site, for example, may update its contents every few hours. In a multiprocessor context, multiple microprocessors may have access to the same pool of shared memory. Here, the multiple cache controllers must somehow communicate to know when memory has been modified so that the individual caches can update themselves and maintain coherency.

Determining the optimal size of a cache so that its performance improvement merits its cost has been the subject of much study. Cache performance is highly application dependent and, in general, meaningful performance improvements decline after a certain size threshold, which varies by application. A typical PC runs programs that are not very computationally intensive and that operate on limited sets of data over short time intervals. In short, they exhibit fairly good locality properties. Typical desktop PCs contain 256 kB of L2 cache and a smaller quantity of L1 cache. Computers that