Software-Managed Caches: Architectural Support for Real-Time Embedded Systems

Bruce Jacob
Electrical Engineering Department
University of Maryland, College Park
Email: blj@eng.umd.edu

The problem with traditional caches

It has long been recognized that, for good performance, applications require fast access to their data and instructions. Accordingly, general-purpose processors have offered caches to speed up computations in general-purpose applications. Caches hold only a small fraction of a program's total data or instructions, but they are designed to retain the most important items, so that at any given moment it is likely the cache holds the desired item. Cache designs work on a relatively simple principle—at any given moment, a program is likely to access data that it has accessed in the recent past, or data that is nearby data that it has accessed in the recent past—and this allows one to build correspondingly simple hardware controllers that achieve significant performance boosts for general-purpose applications. However, hardware-managed caching has been found to be detrimental to real-time applications, and as a result, real-time applications often disable any hardware caches on the processor.

Why is this so? The emphasis in general-purpose systems is typically speed, which is related to the *average-case* behavior of a system. In contrast, real-time designers are concerned with the accuracy and reliability of a system, which are related to the *worst-case* behavior of a system. When a real-time system is controlling critical equipment, execution time must lie within predesigned constraints, without fail. Variability in execution time is completely unacceptable when the function is a critical component, such as in the flight control system of an airplane or the antilock brake system of an automobile.

The problem with using traditional hardware-managed caches in real-time systems is that they provide a probabilistic performance boost; a cache may or may not contain the desired data at any given time. If the data is present in the cache, access is very fast. If the data is *not* present in the cache, access is very slow. Typically, the first time a memory item is requested, it is not in the cache. Further accesses to the item are likely to find the data in the cache, therefore access will be fast. However, later memory requests to other locations might displace this item from the cache. Analysis that guarantees when a particular item will or will not be in the cache has proven difficult, so many real-time systems simply disable caching to enable schedulability analysis based on worst-case execution time.

One solution is to pin down lines in the cache, for hardware systems that support it. System software can load data and instructions into the cache and instruct the cache to disable their replacement. The chief disadvantage of this approach is that it is not amenable to dynamic reorganization; once data and instructions have been pinned, it is often more overhead than it is worth to reorganize the cache contents. What is needed is a flexible, low-overhead mechanism that allows data and instructions to be pinned, and that also allows the contents of the cache to change quickly, under the supervision of the operating system. Software-managed caches allow such behavior, as they determine cacheability based on the reference address.

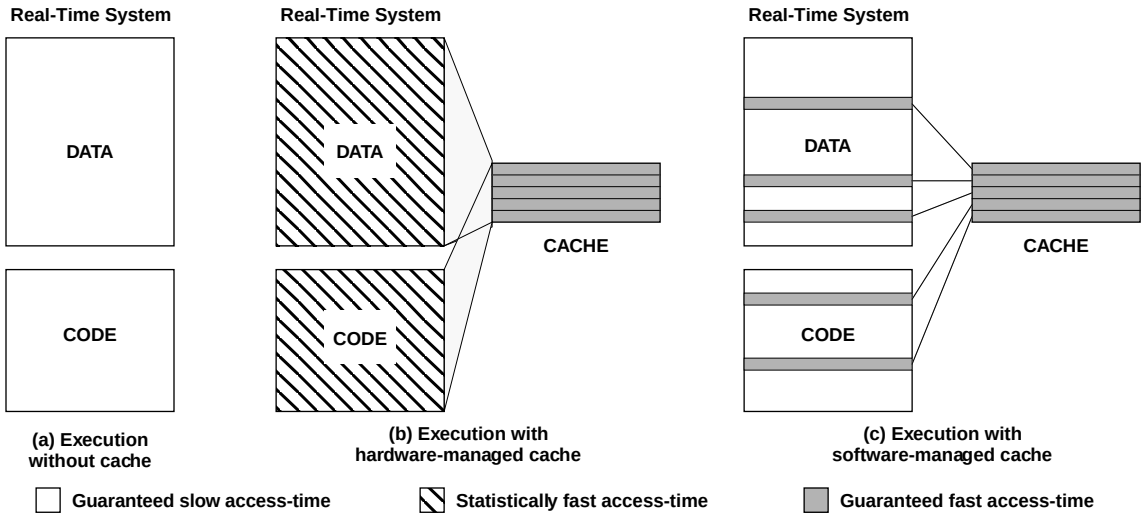

Figure 1. The use of software-managed caches in a real-time system.

Software-managed caches

Software-managed caches allow an operating system to determine on a cacheline-by-cacheline basis whether or not to cache data, and are especially valuable in real-time systems. For example, initialization code of a real-time process would never be cached, while the periodic body of the code would always be cached. Since initialization code only executes once, the loss in performance by not caching the code is amortized over a long execution time. The periodic loop, however, is cached, and results in significantly increased performance during the entire execution, since an RTOS managing the cache can guarantee that the code remains cached for the lifetime of the process.

Figure 1 illustrates this type of selective caching. Figure 1(a) depicts a typical real-time system that runs without caches; access is slow to every location in the system's address space. Figure 1(b) shows the effect of adding a hardware-managed cache; in the steady-state, each item in the address space has a statistical likelihood of currently existing in the cache—it may or may not be in the cache at any given point in time. Figure 1(c) shows the effect of adding a software-managed cache; the software determines what can and cannot be cached, therefore the software can ensure (if so desired) that certain portions of the address space will always be cached. As compared to a traditional hardware-managed cache, timing analysis is as simple as in the non-cached case, because access to any specific memory is consistent, either always in cache, or never in cache. Compared to a processor with no cache, selected data accesses and instructions execute 10-100 times faster.

Work in software-managed caches

At the University of Maryland, we are developing hardware and software models for software-managed caches, as well as compiler techniques for exploiting their capabilities. For more details on the hardware design, see:

Bruce L Jacob and Trevor N Mudge. "Software-managed address translation." *Proc. Third International Symposium on High Performance Computer Architecture (HPCA-3)*, pp. 156-167. San Antonio TX, February 1997.

Bruce L Jacob and Trevor N Mudge. "A look at several memory management units, TLB-refill mechanisms, and page table organizations." *Proc. Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-8)*, pp. 295-306. San Jose CA, October 1998.