ISSN 1392-2114 ULTRAGARSAS, Nr.2(39). 2001.

Caches in DSP processors

M. Genutis, E. Kazanavičius, O. Olsen

KTU DSP Laboratory

Communication Department, Aalborg University

Introduction

While caches are familiar in RISC microprocessors, they've only recently entered the world of DSPs. DSP processor vendors preferred no cache at all or simple instruction buffers instead of complicated memories with caches. Caches bring a fair amount of unpredictability into hardware systems. That was the main reason why caches haven't been used in DSP processors. As DSP processors become more powerful, there appears a need to improve memory system.

The use of caches in DSP processors can be motivated in part by cost. Assuming a reasonable locality of reference in an application, a relatively small amount of cache memory can approximate the performance of a much larger local memory at a significantly lower cost. Thus, cheaper DSPs become cost-effective solutions for a much wider range of applications.

There is one more important advantage of a program cache – it often leads to lower core-processor power consumption. This is because program memory cache is typically located close to the core and, therefore, does not use the large address and data buses that go to the main system memory.

A. The Principle of Locality

A program does not access all of its code or data at once with equal probability. Having recently accessed information in cache increases the probability of finding information locally without having to access memory. The *principle of locality* states that the CPU accesses a relatively small portion of the address space at any instant of time.

The principle of locality has two components: *spatial* and *temporal*. Spatial locality states that for a given memory access, the next access is likely to be sequential. That is, programs reference items whose addresses are close to other recently accessed items. For example, accesses to elements of an array or record show a natural spatial locality. Caching takes advantage of spatial locality by moving blocks from memory into cache and closer to the processor. References to the next location are sometimes separated into a third aspect, known as *sequential locality*.

Temporal locality states that once memory is accessed, the same data or instructions are likely to be used again. That is, programs tend to reuse recently accessed items. Temporal locality is found in instruction loops, data stacks and variable accesses. For example, when executing an instruction in a loop, the loop may iterate many times. Each time the same instructions are executed.

Different programs have different locality degrees. Algorithms that have a lot of control code are not as predictable as those, which iterate in small loops. However, there is no consistent way of measuring the degree of locality of a program. We can only state that one particular algorithm has a higher locality than the other one. We can prove this "empirically" by running these programs on a particular cache configuration and measure their cache hit ratio.

B. Cache Overview

Cache memory is a small, fast memory unit between the CPU and the main storage memory. Cache typically stores the most recently used instructions and data. It makes use of the principle of locality. Cache memory is important; it bridges the gap in capabilities between the CPU and main memory. It allows a small high-speed memory to be effective by storing only a subset of the main memory. The concerns are:

1. Maintaining coherence between cache and main memory.
2. The criteria needed to refresh cache memory as the program is executed.

Two cache architectures are used:

1. Harvard, where instructions and data caches are separate.
2. Von Neumann, where they are unified.

Within the cache memory there must be a way for the cache to know where the necessary information resides. This enforces a mapping on the data/instruction blocks in the cache. There are three general formats for the mapping of a block to the cache:

1. Fully associative.
2. Direct mapped.
3. Set associative.

For an in-depth cache architecture overview the reader should consult 10.

Concepts and quantitative issues

Here we'll take a look at the concepts and numbers, which describe the behavior of the cache.

The *memory access time* is the time between the submission of a memory request and the completion of transfer of information into or from the addressed location. The *memory cycle time* is the minimum time that must elapse

between two successive operations to access locations in the memory (read or write).

A cache *hit* occurs when the requested data are found in the cache, otherwise a *miss* occurs. Thus we can define the cache *hit ratio* as:

$h = n_h / n$, where n_h – number of cache hits, n – number of overall memory accesses.

Similarly, cache miss ratio:

$m = n_m / n$

$h = 1 - n_h / n$, here n_m – number of cache misses.

Assuming t_c – cache access time and t_m – main memory access time, memory access time in case of a cache miss would be:

$ta = tc + tm$

Suppose the reference is repeated n times and, after the first reference, the location is always found in the cache. Then the average access time equals:

$$t_{av} = (nt_c + t_m)/n = t_c + t_m/n = t_c + (1-h)t_m$$

The concepts and equations above are the basic measures used in describing cache behavior.

Cache performance

Cache hit ratio depends very heavily on the programs being executed and the overall workload. The exact value of miss ratio cannot be found for any particular computer system. The miss ratio also depends upon the cache organization chosen, the size of the internal division of the cache, the write policy and the replacement algorithm.

There are three basic methods of obtaining an estimate of the miss ratio:

Trace-driven simulation – this method is probably the most popular. In this method, programs are selected for execution on a computer system not necessarily having a cache. A record of the instruction and data references is kept. The processor trace facility is generally used. Specific cache organizations are then simulated, using the instruction/data references that have been gathered, to determine the miss ratio.

Direct measurement – the memory reference sequence could be obtained by direct measurement by attaching special monitoring hardware to the computer system to record the memory references.

Mathematical modeling – models can be developed based upon differential equations, statistical and probabilistic techniques. After a mathematical model has been obtained, it is generally compared to experimental simulation results.

Real-time constraints and cache

Real-time systems are commonly considered to be used only for specific applications. Nowadays real-time systems are being built not only for life-critical applications, where the cost factor is of second importance. Emerging areas like car computers, multimedia computing, gesture recognition, voice interaction and the like demand real-time capabilities, but at a relative low cost. These applications require cheap hardware platforms to be competitive in the marketplace.

It is highly desirable to utilize cache memory in real-time systems, provided that the behavior of cache can be controlled in deterministic manner. A major reason for the absence of caches in real-time applications is due to lack of understanding of its predictability. This refers to the ability to place a tight bound on the worst-case execution time (WCET) of the task.

DSP cache specifics

According to [5], two factors reduce the utility of caches in DSP applications:

1. Data access patterns.
2. Real-time constraints.

DSP applications tend to process large amounts of data. Often, there is less locality in data accesses in DSP applications than in other types of applications. Because of that data caches are less effective for DSP applications. However, DSP applications display very strong locality in instruction accesses. DSP application execution time is typically dominated by a small number of small loops, making instructions caches appear attractive.

Dynamic behavior of caches makes them difficult to apply in real-time DSP applications. The programmer, implementing a DSP application, doesn't know what will be in the cache at the moment his program will be executed.

Some caches provide features to help programmers design real-time applications. For example, the ability to pre-load and lock portions of the caches. Using this feature, portions of the caches can be loaded with specified blocks of instructions or data, and then locked so that the loaded instructions or data cannot be displaced from the caches. Once portions of the caches have been locked, these portions act as a fast local memory, and access times for data and instructions in the locked portions become deterministic. However, locking portions of the caches reduces the effectiveness of the caches for the remaining instructions and data by reducing the size of the caches available for the remaining instructions and data.

DSP processors have traditionally avoided caches, relying instead on small amounts of on-chip memory. This memory is often treated as a manually controlled cache. Developers explicitly transfer instructions and data between on-chip and off-chip memory. DSP processor caches usually differ from those found in general-purpose caches. As noted in [6], the key differences between general-purpose processor caches and those of DSP are:

1. DSP processors use instruction caches (or no caches at all) but very rarely use data caches.
2. DSP processor instruction caches are generally much smaller than general-purpose processor instruction caches. They are often integrated into the processor core itself, as a part of the processor's control unit.

3. DSP processor instruction caches are usually supplemented by additional on-chip non-cache SRAM.
4. DSP processor instruction caches are usually simpler in their organization and operation than general-purpose processor caches, and as a result their impact on execution time is usually more predictable.
5. DSP processor instruction caches are more specialized than general-purpose processor caches. Often they are used only in conjunction with special instructions designed to provide low-overhead looping constructs.

DSP Processor cache case studies

C. Texas Instruments C6211/C6711 Caches

C6211 is a fixed-point and C6711 is a floating-point DSP processor. They do not differ in their memory architectures. Further I'll talk of both processors while mentioning only the C6211 device.

C6211 provides two level memory architecture for the internal program and data busses. The first level memory for both the internal program and data bus is a 4K-byte cache, designated L1P for the program cache and L1D for the data cache. The second level memory is a 64Kbyte memory block that is shared by both the program and data memory busses, designated L2.

Table 1: TMS320C6211 Cache architecture

Cache Space	Size (Bytes)	Associativity	Line Size (Bytes)
L1P	4K	Direct	64
L1D	4K	2-way	32
L2	64K	1- to 4-way	128

1) Level-one Data Cache (L1D)

The L1D is organized as a 2-way set associative cache with a 32-byte line size. Set associative cache provides additional flexibility to a direct mapped cache. This cache architecture is beneficial for DSP data, which tends to be more random and have larger strides than program data.

Each way in the L1D caches 2Kbytes of data. 2-way set associative cache reduces the chance of a cache thrash since two thrashing addresses can be stored in the cache simultaneously. This is a beneficial architecture for DSP data, which often accesses multiple arrays simultaneously, such as arrays of coefficients and samples. A 2-way set associative cache is an advantageous design since the CPU has two data paths, which could simultaneously access two different data arrays. The L1D minimizes the chance of these two data paths thrashing.

The L1D replaces data with a Least Recently Used (LRU) replacement strategy. LRU replacement chooses which set to update with new data by determining which of the two cache ways was accessed least recently. The new data is then placed in the appropriate set of that least recently used way. LRU is the best replacement strategy for set associative caches because of the temporal locality of data – once data has been used it will be probably needed again within a short time.

2) Level-one Program Cache (L1P)

The L1P is organized as a direct mapped cache with a 64-byte line size. A direct mapped cache is well suited for DSP algorithms, which tend to consist of small, tight loops that rarely thrash. The L1P line size provides a modest prefetch of the next fetch packet, eliminating the startup latency for fetching that packet.

In direct mapped cache every cacheable memory location maps to only one location in the cache. Thus, the cache controller needs to check only one location in the tag RAM to determine if requested data is available in the cache. DSP algorithms primarily consist of loops that execute the same program kernel many times on multiple data locations. Such algorithms remain in a loop for a long before proceeding to the next kernel. The L1P is large enough to hold several typical DSP kernels simultaneously. Since these kernels execute sequentially, they will not thrash in the L1P. Thus, a simple direct mapped cache is all that is needed to achieve considerable program performance without requiring complex caching hardware.

When a cache miss occurs, the L1P requests an entire line of data from the L2. In other words, both the requested fetch packet and the next fetch packet in memory are loaded into the cache. Since most applications execute sequential instructions, there is a high likelihood that the next fetch packet will be immediately available when the CPU requests it. Thus, the startup latency to fetch the next fetch is eliminated by eliminating the startup latency and reducing misses reduces the execution time of an application considerably compared to a cache with a smaller line size.

3) Level-two Cache/Unified Memory (L2)

The L2 is a 64-Kbyte SRAM divided into four 16-Kbyte blocks. The L2 is a unified memory, used for both program and data. The amount of program or data in the L2 is configurable. For example, if the application requires only 7 Kbytes of program space and 57 Kbytes of data space then both could be linked into the L2 at the same time. Likewise, if the application needed more program space than the data, the majority of the L2 RAM could be linked as program space.

Each of the four blocks can be independently configured as either cache or memory mapped RAM. This allows dictating the amount of the L2 that is used as cache and how much is used as RAM. If the application uses some data, which must be accessed quickly, that data can be linked into an L2 block, which is configured as RAM. The rest of the L2 can be configured as cache, which will provide high performance operation of the remaining program and data.

By unifying the program and data in the L2 space, the L2 cache is more likely to hold the memory requested by the CPU. It enables the on-chip memory to contain more data

than program when highly computational, looping code is being run to process large data streams. For long, serial code with few data accesses, the L2 may be more densely populated with program instructions. The unification allows allocating the appropriate amount of memory for both program and data and keeps the on-chip memory full of instructions and data that are most likely to be requested by the CPU.

Texas Instruments have adopted a new cache scheme in DSP. Such scheme has never been tried before because of the indeterminacy that is brought by the L1/L2 cache levels. DSP applications increase in size and processor manufacturers cannot provide enough on-chip memory to place these applications. Thus, two-level caches might become more popular in the future.

D. Motorola DSP56300 Cache

The Motorola's DSP56300 device has only the instruction cache. When enabled, it comprises 1024 24-bit words of program memory. The purpose of this user-transparent instruction cache is to lower the requirements for expensive, high-speed memory.

The cache is 8-way fully associative. Least recently used algorithm is used for sector replacement.

Programmer can lock each cache sector (out of the 8, each consisting of 128 24-bit words). Locking a sector prevents its replacement in case of a miss even if it would have been its turn to be replaced.

The cache can be flushed with one instruction (for example, for task switching).

The penalty incurred for a cache miss is identical with the one for a regular instruction fetch from external memory.

Conclusions

Caches in DSP processors appeared not a long time ago. Their use have been delayed mostly because of their probabilistic performance increase which they bring to DSP applications. It's difficult to meet real-time constraints when using caches. First, there appeared simple instruction buffers, then instruction and data caches. *Texas Instruments* came up with a two level cache architecture. The use of caches is mostly motivated by cost – large on-chip memory is expensive. As DSP applications become larger, we expect more DSP processors incorporate caches. Processor manufacturers leave freedom to programmers – caches can be disabled, cache sectors can be locked. Further, the author of this report is going to benchmark the C6211 two-level cache with DSP application benchmarks such as GSM-EFR encoder/decoder, Viterbi.

References

1. **Basumallick Swagato and Nilsen Kelvin.** Cache Issues in Real-Time Systems. Technical Report, Department of Computer Science, Iowa State University, May 1994.
2. **Bordeaux Ethan**. Advanced DSP Performance Complicates Memory Architectures in Wireless Designs. Wireless Systems Design. April 2000. P. 20-24.
3. **Dropsho Steve.** Real-Time Penalties in RISC Processing. Technical Report, Department of Computer Science, University of Massachusetts-Amherst, December 1995.
4. **Jeremiassen Tor.** A DSP with Caches – A Study of the GSM-EFR Codec on the TI C6211. In International Conference on Computer Design, (ICCD '99). October 1999. P. 138-145.
5. **Lapsley Phil, Bier Jeff, Shoham Amit and Lee Edward.** DSP Processor Fundamentals: Architectures and Features. IEEE Press. 1997.
6. DSP5630 Family Manual. Motorola, November 2000.
7. **Simar Ray.** Two-level cache quickens pace for DSP performance. Embedded Systems Development. February 2000.
8. TMS320C6000 Peripherals Reference Guide. Texas Instruments. April 1999.
9. TMS320C6211 Cache Analysis. Texas Instruments. September 1998.
10. **Wilkinson Barry.** Computer Architecture: Design and Performance. Prentice Hall. 1996.

M. Genutis, E. Kazanavičius, O. Olsen

DSP spartinančiųjų atminčių panaudojimas signalų procesoriuose

Reziumė

Darbo tikslas – apžvelgti spartinančios atminties panaudojimo DSP sistemoje efektyvumą.

Darbe aptariami pagrindiniai atminties panaudojimo principai ir koncepcija. Analizuojama ekfektyvumo klausimai. Palyginta kelių firmų DSP sistemos.

Darbas atliktas bendradarbiaujant su Danijos Aalborgo universitetu.

Pateikta spaudai 2001 04 23

DOI: 10.5755/j01.u.39.2.8051