(Print)

JCIT Vol. 9(5), 55-59 (2018). Periodicity-2-Monthly

(Online)

JOURNAL OF COMPUTER & INFORMATION TECHNOLOGY
An International Open Free Access Peer Reviewed Research Journal of Computer
Science Engineering & Information Technology
website:- www.compitjournal.org

Performance improvement by Software controlled Cache Architecture

ISHAN KUMAR

ASIC Engineer Nvidia Corporation Bangalore (India)

Corresponding Author Email:- ishank4@gmail.com

<http://dx.doi.org/10.22147/jucit/090502>

Acceptance Date 20th September 2018

Online Publication Date 2nd October, 2018

Abstract

In advanced multimedia communication based systems, performance improvement is one of the most important issues. In communication applications which are data intensive, major portion of processor power is consumed by data caches. The cache architecture cannot be taken care of specifically for an application in case of an integrated communication system. As a result, a big amount of cache memory is not used. In the following paper, a software-controlled cache architecture has been proposed, which improves the energy efficiency of the shared cache in an integrated communication based system. Data types are allocated for an application for different cache regions. Only the allocated cache regions are activated. The effectiveness of software-controlled cache after integration is tested in a communication based System on chip. The results show the performance improvement of the system on chip up to a huge level on ARM-like cache architecture.

Key words : performance improvement, cache memory, energy efficiency, software cache, address buffer, translation, page access.

I. Introduction

For better performance, applications need fast access to their data and instruction sets. Accordingly, general-purpose processors have caches for fast computations in generalpurpose applications. Caches contain a small fraction of a program's whole data or instruction set nly designed to hold the most important items so that cache can retain the desired item at any significant point of time. Cache is based on simple principle *i.e.* at any particular moment, a program is supposed to access real time data which has been accessed in past. This allows building simple hardware controllers that achieve significant performance improvement for general

applications. Most times, the caching has been proved disastrous in real-time applications. Hence real-time applications disable most of the hardware caches on processor. General-purpose systems expect high speed, which is related to the behavior of a system. Accuracy and performance of a system are also very important factors as well. Generally execution time must be within the limit given by constraints. Variability in execution time is not acceptable for critical functions in flight control system of an airplane or the antilock braking system in an automobile.

The main issue with the traditional hardware-managed caches in systems is that a cache might not

contain the expected data at any specific time, though they allow having a good performance improvement of the system. Access becomes faster when data is present in the cache. If the data is not present in the cache, access becomes quite slow. Typically, for the first time when a memory item is generally requested, it might not be present in the cache. Further continuous accesses for the item are likely to find it in the cache, so access becomes faster. But memory requests on other areas might move this item from cache. Analysis of the presence of an item in the cache has become very difficult in recent times. So, disabling caching is a good option in many real time systems to enable analysis based on the worst-case execution time. To pin down the lines in the cache might be a solution for hardware which supports it. System software has the option to load data and instructions into cache and command the cache for disabling their replacement. Once data and instructions are pinned, it is not susceptible for dynamic identification which is the major disadvantage of this approach. A flexible, low-overhead mechanism is always expected which allows data and instructions to be pinned, and allows the contents of cache to change faster as well.

II. Common Caches Available

Cache is generally used to increase the accesses to storage devices mainly in tape, disk drives, and memory units. The principle on which it works is the principle of locality of reference *i.e.* applications to reference a small amount of data within a stipulated time. Storage device is built by a technology that has a fixed access time and cost, where faster technologies will have a lower access time and more cost per storage unit compared to slower technologies. A cache for a given storage device is built from a technology which is faster than that of the storage device in discussion and only needs to be quite large to hold the working set of the application. The set of data items and instructions of the application are used to perform its computations for effectiveness. Most of the application accesses are generally satisfied by the cache, and most of the time the overall access time will be that of the cache: far faster than the larger storage.

A. Fundamental Cache Operations :

The data and the tags are the main two parts of a general cache. Generally, when a cache is smaller than an entire address space, there is a chance that any specifically requested data is not available in the cache. Some mechanism to determine whether any particular data is present in cache should be there. The tags are a list of valid entries in cache present for this purpose, one per data entry. Hence every tag entry can identify the contents of

its related data entry. Virtually, most of the hardware caches operate in this way, one indexes the cache and the associated tag entry indicates the data stored in the cache. When the tag matches, *i.e.* if it corresponds to the requested data, then the data in the data entry is read out. Cache lookup procedure can be divided into 2 parts: fully associative and direct-mapped. In fully associative lookup, the cache is a small hardware database. Data can be placed in the cache irrespective of any location; the tag field can identify the data contents. A search checks the tag of every data saved in the cache. If any tag matches the tag of the requested address, then it is a cache hit: the cache contains requested data. In a direct-mapped lookup organization, a given data can only exist in one cache entry, usually determined by a *su* address though the most common index is low-order bits of tag field.

B. Basic Cache Architecture :

Cache tags and cache data are the two integral parts of any cache. Individual data entry is termed as cache block or cache line. The tag entries generally identify the contents of their corresponding data entry.

C. Associative Lookup Operation :

Fully associative lookup structure is also called as Content-addressable memory (CAM). Any entry that has the same tag as the lookup address matches irrespective of its location in the cache. This structure helps in reducing cache contention but look up table (LUT) is expensive as the tag of every entry is matched with the lookup table address. The lower bits of the address not used in tag match determine what portion of the cache line to be sent to the requestor after the line is read. In associative scheme there are number of tag matches, where *n* is the no. of cache line. Here only one tag match is acceptable because the requested data can only be found in any one location: Directly mapped cache is extremely fast to search as there can only be one place for any particular data. A set associative cache is intended for fast lookup and lower contention¹.

D. Cache Architecture Organization :

Several cache organizations are generally available based on the cache indexing and tag information. The cache is a limited database, and the address of the physical or virtual data, corresponds to the key of the database. Most of the instruction and data caches are mapped directly. For specialized cache structures full associativity is reserved². In directly mapped caches, a small portion of the key can be used for choosing a data set. Only one cache line exists at a given index in case of directly mapped caches. More than one cache lines exist in case of set-associative

caches. If any one of the tags matches the key, the specific cache line is read out as the key might be a virtual or a physical address.

i) Physically Indexed, Physically Tagged Architecture:

In this type of cache design, the virtual address must be translated before the cache can be accessed as the cache is indexed and tagged with its physical address location. The cache can be completely controlled by hardware in this case and the operating system is not responsible of cache management. Disadvantage of this design is that address translation is in the critical path. When clock speed increases, this becomes an issue. As the memory size increases, and

Fig. 1 Physically indexed, physically tagged architecture

ii) Virtually indexed, virtually tagged architecture :

In this mechanism, the cache is indexed and tagged by virtual Address. Address translation is not required throughout this process and that's the main advantage. Translation buffer is not needed and if used, that only needs to be accessed while a requested data is absent in the cache location. On a cache miss, the virtual address should be used to load the data from physical memory location. A TLB can speed up the process if mapping is present in TLB. Size of the TLB can be bigger if it's not in critical path: but this will result in slower access time, a larger TLB contains lots of mapping information. The virtually indexed, virtually tagged organization is illustrated in Fig. 2.

E. The advantage and disadvantage of caching :

Cache contains important data close to the processor which helps in faster processing of data. Advantage of placement is gained in 2 ways: While referencing an item, it is placed inside the cache so if the item is once again referenced in later time, it might have the possibility to be present inside the cache, as a result access becomes faster. Main intention of a program is to reuse data in later stages. When a data is considered, the nearby data is also brought into cache. A cache line is generally larger than a single data. So, if a program uses data near the

original data, the data should be present in the cache. Generally, a program tries to reference items that are placed in nearby locations and it has used in the past. Usually, the data found in the cache: items are rarely replaced immediately, and thus programs can have good performance in case of re-reference of data and instruction

The cache contains the most important instructions and any reference to an object which is not present in the cache. The execution path of a typical program is based on input data which is really difficult to predict.

The problem with this arrangement is that, in the steady-state, the cache is full of important data, and any reference to an object that is not already in the cache displaces something that has been referenced in the past and is therefore important. Because the execution path in a typical program is based partly on the input data, it is difficult to predict exactly which instruction

Data reference pattern of a program is based on the data values; thus it is not feasible to predict exactly about the content program will need in later stages. It is almost impossible to decide about the duration of data that will last in cache before it is replaced by other data. Its very tough to predict the steady cache contents and therefore it is very difficult to detect the execution time of any instruction. In real time systems, precision is one of the major issues³.

III. Software Controlled Cache Development :

Mainly two primary cache organizations exist for real-time processing. The first one is used in digital signal processors. Usage of on-chip SRAMs forms a separate space from main memory. Instructions and data generally appear in these memories when software relocates them there explicitly.

Software-managed virtual cache has made the transition to real-time embedded systems in recent times. Software-managed cache organizations generally allow the software to determine on a cache line basis, whether or not to move cache instructions and data which are especially very important in real-time systems. For example, initialization code of real-time systems would never be cached on priority wise but the periodic body of the

code is always cached on a higher priority. The loss of performance for not caching the code is encountered over a long execution time as the initialization.

Fig. 2. Virtually indexed, virtually tagged architecture code executes only once. However, periodic loop is mainly cached and it results in increased performance while entire execution process⁴.

A. Separate Space for SRAM :

Software execution sees a namespace that spans several different storage types. The software is thus completely aware of the storage types available and can make intelligent choices regarding the space where each function or data object should reside for best performance. This specific memory map contains two on-chip SRAM arrays available in the low region of the address space. On top of the address space, DRAM array exists, which is located off-chip in this example. Middle of the memory map contains the devices and the ROM array as shown in Table 1. Considering, Memory areas have the following sizes and correspond to the following ranges in the address space⁵.

Address location	Size (Kbytes)	Device
0x0000 0x0FFF	4 KBytes	SRAM-0
0x1000 0x1FFF	4 KBytes	SRAM-1
0x2000 0x3FFF	8 KBytes	invalid
0x6000 0x6FFF	4 KBytes	invalid
0x4000 0x5FFF	8 KBytes	ROM
0x8000 0xFFFF	32 KBytes	RAM

The static value of the function can't be used to call the specified function. Future invocations of functions have to use the address 0x1000 instead of the function which is located in ROM, otherwise those invocations have to call the ROM version of the same function but not the cached one. This software-managed cache organization works very efficiently.

In embedded systems, address space protection

is a future issue. For address space protection, providing access to the memory arrays via a virtual memory mechanism used in general-purpose system, a memory-management unit and a translation buffer is quite efficient. The arrangement requires a little different from general-purpose systems. Firstly, the management of memory system becomes more complex because there is more than simply a DRAM array. The management of the TLB should be more deterministic than it is in typical general-purpose systems where random replacement techniques are used.⁶

For memory-management interface, a DSP-based operating system has to provide several changes on malloc() function, each of which allocates a virtual region to the process which maps to a separate area of the namespace. As an example a set of functions that a DSP based OS can export have been mentioned below:

```

void *sram0_malloc( size_t size );
void *rom_malloc( addr_t start, size_t size );
void *sram1_malloc( size_t size );

```

void *dram_malloc(size_t size) rom_malloc() function allocates a region within the process address space which is mapped to a region in ROM array. rom_malloc() function requires the software to specify a region in the storage device. This memory-allocation interface should have the desired effect of allow a process to make device-specific optimizations. It protects the virtual address space of processes as well.⁷

B. Software Managed Virtual Caches :

When software gets involved in cache-fill and decouple, the translation hardware then the software gets the control of the memory system. In TLB and in MMU software has the control over memory system. In general purpose systems basically hardware handles cache-fill and decouple but if a memory access misses the cache, hardware gets ready to fetch data from memory. If virtual address translation hardware is used, the hardware should translate address before it is sent to the main memory.

Software is responsible for all the activities related to cache fill by the use of upcalls to the software that occurs on cache misses. Cache miss interrupt software and vector to handler which fetches the referenced data and places it inside cache. Software fetches data but doesn't place it inside the cache if it determines not to cache it. For avoidance of frequent interruptions for accessing data which is not cached, a separate region of the virtual space relates directly with the physical memory, hardware must identify addresses in this region as physical and non cached. Those addresses are send directly to DRAM

array.⁸

It works in general-purpose style which caches the datacopies placed in DRAM system. If a real time system works without caches then processing time is too slow. By adding a hardware-managed cache, each item in address space might exist in the cache. A software controlled cache determines what should be cached and it tells that certain part of address space has to be cached all the time. Timing analysis is also simple as in hardware controlled cache because access to any specific memory location is always consistent. A processor without cache executes instructions 10-100 times slower compared to this. Address translation hardware is also an option in this mechanism. It speeds up the translation of addresses which miss the cache, and can be used to create a memory access beyond cached and uncached.

IV. Conclusion

This paper focuses on several efficient software-oriented cache management schemes for real-time systems by make use of SRAM caches present on-chip. The paper covers general-purpose caches with DSP caches which share the same namespace similar to DRAM systems. Here, address-space protection is provided by a virtual memory technique as in case of general-purpose systems. Implementation of a virtual memory layer and address translation adds an extra overhead that can be absent if user level processes are executed directly on top of memory system. If SOC's are increasing in general purpose systems, with software developed by multiple vendors in that case, this performance overhead is a small burden almost negligible in comparison with process protection and modularity.

Acknowledgement

The author would like to thank Nvidia Graphics for providing the necessary financial assistance and required guidance.

References

1. N. P. Jouppi, "Cache write policies and performance" In Proc. 20th Annual International Symposium on Computer Architecture (ISCA-20), May 1993, pp. 191 -201.
2. B.L. Jacob and T.N. Mudge, "A look at several memory-management units, TLB-refill mechanisms , and page table organisations." In Proc. Eighth Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS- 8), San Jose, CA , October 1998, pp 295-306.
3. P.J. Denning. "Working Sets Past and Present." IEEE Transactions on Software Engineering , vol. 6, no. 1, pp 64-84, January 1980.
4. B.L. Jacob, "Software-managed caches: Architectural support for real-time enabled systems." In CASE958: Workshop on Compiler and Architecture Support for Embedded Systems, Washington DC.
5. B.L. Jacob and T.N. Mudge , "Software managed address translation." In Proc. Third International Symposium on High Performance Computer Architecture (HPCA-3)
6. Qian Yu, "Characterizing the Rate memory Trade-off in Cache network within a factor of 2", IEEE 2018.
7. Jun Shiomi , " Maximizing energy efficiency of on chip caches Exploiting Hybrid Memory Structure", 2018 IEEE Conference on Network Softwarization and Workshops.
8. Liang Feng, "Static and Dynamic Hybrid Cache Management for CPU-FPGA Platforms. 2018 IEEE