INTERNATIONAL JOURNAL OF ADVANCE RESEARCH AND DEVELOPMENT

(Volume 3, Issue 11)

Available online at: www.ijarnd.com

Cache performance improvement using software-based approach

Adarsh Mittal[1], Srishti Chadha[2]

[1]*Research Scholar, Nvidia Graphics Pvt Ltd, Bengaluru, Karnataka*

[2]*Research Scholar, IBM, Bengaluru, Karnataka*

ABSTRACT

Cache usage is a common mechanism for increasing the speed and performance of memory access and are widely used from simple microprocessors to the complex multi core-based designs. It is seen that the cache is not well designed for the embedded systems as the performance is based on probability and is not deterministic. It is difficult to guarantee the time a data will be present or absent in the cache memory. So, it becomes difficult for the embedded system to use the ability of cache to increase the performance. Normally, many real-time systems simply switch off the cache technique and use the scheduler algorithm based on the worst time memory access. There is various software stack that provides the advantage of cache time without the limitation that a hardware-based cache offers.The paper talks about the different organization and operations of cache techniques found in generally used processors, signal processing units and microcontrollers.

Keywords— *Memory access improvement, Efficiency, Hardware-based cache, Software-based cache, Virtual memory access, Translation, Page access*

1. INTRODUCTION

It is desired to have better performance for a system which requires fast access to the data and instruction sets. In order to achieve that general-purpose processors have caches that speed up the computations in general- purpose applications. Cache memory unit contains a small fraction of a program's whole data or instruction set which are explicitly designed to hold the most important items so at a given probability that will store and retain the most desired data. The principle in which cache works is simple i.e. at a given point of time a system is most likely to access the data and information which it has requested in the past. This helps architectures to build simple hardware controllers to improve the performance of an application. In most cases, it is seen that the catching technique is not suited for real-time applications. Hence most of the embedded system applications switch off most of the hardware caches on the processor. The two important factors that control the general-purpose systems is the accuracy and performance of a system. Generally, it is desired that execution time must match the constraints as they control the flow of data. Variability or difference in execution time is not acceptable for critical functions in an airplane flight control system or the system designed for the antilock braking system in an automobile.

The common problem with the traditional hardware-based caches in real-time systems is that there is a high probability that the cache may not contain the desired data at any specific time or moment, although it may allow a performance improvement in the system. Access becomes faster when data is available in the cache. If the data is not available in the cache, access becomes quite slow and difficult. Typically, for the first time when the desired item is generally requested, it might not be present in the cache. Further with the continuous accesses for that particular item are likely to find it in the cache, so access becomes faster. But since the different memory requests on might move this item from cache it is likely that the important data is replaced in the cache. Study of the presence of an item in the cache has become very difficult in recent times. So, much real-time application disables caching technique to enable analysis based on the worst-case execution time. To store the lines in the cache might be a solution for hardware which supports it. System software has the functionality to load data and instructions into the cache and command the cache for disabling their replacement so that the data is available. Once data and instructions are pinned to the cache, it is not susceptible to dynamic identification which is the major drawback of this approach. A flexible mechanism or application is expected which allows instructions to be pinned so that they are not prone to changes.

2. COMMON MEMORY CACHE

Cache is generally used to increase the performance of access to storage devices mainly in disk drives, tape, and sub-memory units. The principle of working is based on the locality of reference i.e. any applications to reference a small amount of data within a fixed interval of time. The devices which are built by a technology that has a fixed time of access and cost, where any fast technologies will have a slower access time and high cost per storage unit compared to slower technologies. A cache memory system is built from a technology which is normally faster than that of usual storage devices and only needs to be large enough to hold the working set of the instructions that an application needs. These are the set of data items and instructions of the application are used to perform its computations for performing a task.

2.1 Fundamental cache operations

There are two important parts of a general cache - data and the tags. Usually, when a cache is small in size than an address space, there is a high chance that the particular requested data is not accessible or stored in the cache. There is a mechanism to find out whether any particular data is available in cache or not. The tags, in particular, are a list of valid entries in the cache

which are available for this purpose, each one is associated per data entry. Hence every tag entry can be used to identify the contents of its related data entry. Virtually, many of the hardware memory caches operate in this way, one indicates the cache and the associated tag entry indicates the information stored in the cache. Whenever a tag matches, i.e. if it corresponds to the valid requested data, then the data in the table of data entry is read out. Cache lookup concept can be divided into 2 parts: fully associative mechanism and direct-mapped mechanism. In the fully associative lookup, the cache hardware is small. Data can be put in the cache without any bound of location; the available tag field can identify the content of the data. A simple search algorithm checks the tag of every data available in the cache. If any data matches the tag of the requested address, then it is called cache hit which means the cache contains the requested data. In a direct-mapped lookup system, a given data can only be available in one cache entry which is usually determined by a subset address although it has the most common index as a low-order bit of tag field.

2.2 Basic Cache Architecture

The two integral part of a cache is Cache tags and cache data. Individual data entry in a memory system is termed as cache block or cache line. The tag data entries generally identify the contents of their corresponding data entry.

2.3 Associative Lookup Operation

A fully associative lookup operation is commonly named as Content-addressable memory (CAM). Any entry which has the same tag as the lookup address matches irrespective of its position in the cache system. This method helps in reducing cache contention but look up table (LUT) is large and hence become expensive and difficult to apply as the tag of every entry is checked for a match with the lookup table address. The lower bits of the address data which are not the part of tag match determine what location of the cache line to be sent to the system requesting the data. In associative lookup scheme, there is n number of tag matches, where n is the no. of the cache line that is there in the system. In this system only one tag match is acceptable because the data which is requested can only be found in any location: Directly mapped cache is fast to search as there can only be one place for any particular information. A set associative cache is intended for fast lookup based search and lower contention[1].

2.4 Cache Architecture Organization

Different cache organizations are available based on the cache indexing and tag information. The cache organization 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 reserved [2]. Indirectly mapped caches, a small portion of the key can be used for choosing a data set. There exists one cache at a given index in case of directly mapped caches. More than one cache lines or multi-line system exist in case of set-associative caches. If anyone of the tags matches the key, the specific cache line is read out as the key might be a virtual or a physical address.

2.4.1 Physically indexed, physically tagged architecture: In this type of system cache model, the virtual address should be translated before the cache can be accessed as the system cache is always indexed and tagged with its physical address location. The cache can be easily controlled by a hardware-based mechanism in this case and the operating system is free from the responsibility of cache management. The disadvantage of this design is that address translation is in the critical path of the design. When clock speed increases, it becomes an issue.

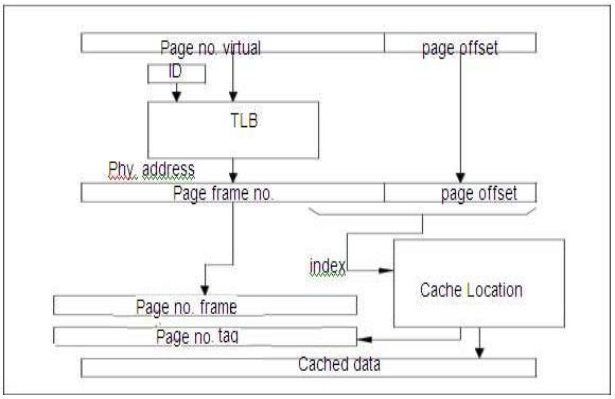


Fig. 1: Physically indexed, physically tagged architecture

2.4.2 Virtually indexed, virtually tagged architecture: In this system, the cache is indexed and tagged by virtual Address rather than real address. So there is no need to address translation in the process and it is the most important advantage. There is no need for Translation buffer and if used, that only needs to be requested while a requested data is absent in the cache location. On a cache miss, the virtual address should be applied to load the data from a physical memory location. The usage of TLB can speed up the process if the mapping is present in TLB. The size of the TLB can be bigger if it's not in a 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 figure 2.

2.5 The advantage and disadvantage of caching

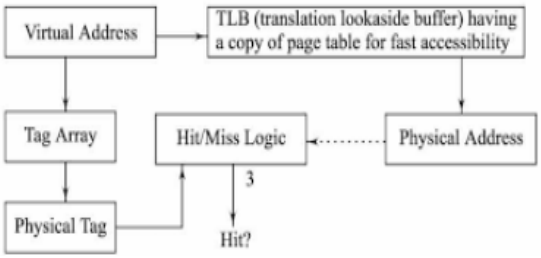
Cache has the most important data close to the processor which helps in increasing the speed of processing of data. The advantage of placement is applied in 2 ways: While when an item is referenced, it is placed inside the cache memory unit so if the item is once again referenced in a later time, it might have the probability to be available inside the cache, as a result, this access the system becomes faster. The main idea of a program is to reuse data in later stages of the pipeline. When a data is picked, the nearby data is also copied into the cache. A cache line is typically larger than a single data set. So, if an application uses data near the original data, the data will be available in the cache. Generally, a program tries to reference items that are placed in nearby locations and it has encountered in the past. Usually, the data found in the cache: items are not usually replaced after one new encounter, and thus programs can have a good efficiency in case of re-reference of data and instruction

The cache system has the most important data an instruction and any reference to an object which is not usually present in the cache. The execution path of a typical program is based on input data which is not an easy task to predict.

The problem with having these set of architecture is that, in the steady-state, the cache is full of important instruction, and any reference to an object that is not already in the cache replaces some important data that has been accessed in the past. Because the execution path in a typical program is based partly on the input data, it is difficult to predict exactly which instruction needs to be referenced.

Since the data reference concept of a program is based on the data values; thus it is not ideal to predict exactly about the

content program will need in later stages. It is almost impossible to decide about the time the particular data that will last in the cache before it is replaced by other data. It is very hard and nondeterministic 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 issues [3].



3. SOFTWARE CONTROLLED CACHE DEVELOPMENT

Usually, two important primary cache architecture exists for real-time processing. The first method is used in digital signal processors. Use SRAMs forms a completely separate space from main memory. Instructions and data which are usually stored appear in these memories when software relocates them there explicitly.

A software-based virtual cache has made the transition to real-time embedded systems in most recent times. Software-based cache system generally allows the software aspect to determine on a cache line basis, whether or not to reallocate the cache instructions and data which are very important and relevant in real-time systems. For example, the initialization part of the code of real-time systems would never be cached on priority but the most reoccurrence body part of the code is always cached on a higher priority. It is seen that the loss of efficiency for not caching the code is encountered during a long execution time as the initialization.

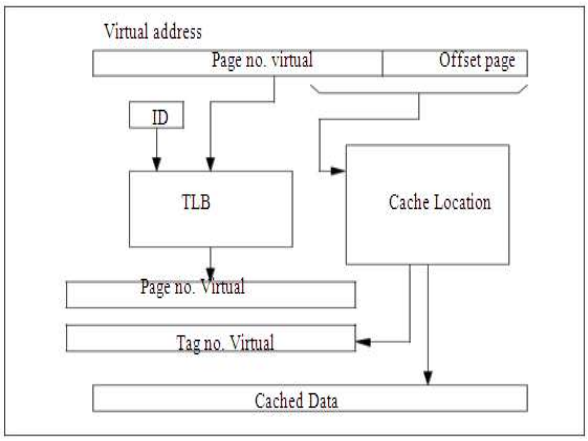


Fig. 2: Virtually indexed, virtually tagged architecture

3.1. Separate Space for SRAM

Software-based execution system views a namespace that ranges several different storage types. This software is completely aware of the different storage types available in the e-market and hence are smart enough to make intelligent choices regarding space where each function or data object should be residing for best performance. This memory map specifically contains two on-chip SRAM arrays available in the low region of the address space. On the very top of the space of address, DRAM array exists, which is usually placed off-chip in this example. The middle part of the memory map contains the peripherals and the ROM array as shown in Table 1. Assuming, Memory system areas have the following sizes and correspond to the following ranges in the address space [5].

Table 1: Memory map with devices and size

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

Here the static value of the function can't be applied to access the specified function. Next set of invocations of functions have to apply the address 0x1000 instead of the function which is located in the system of ROM, otherwise, those invocations have to first access the ROM version of the same function but not the cached one. This software-managed cache system works very fast and efficiently.

In real-time systems, address space protection is a new issue. For the purpose of address space protection, providing access to the system arrays via some virtual memory mechanism used in the real-time system, a memory-management system unit and a translation buffer is quite effective. This arrangement of the system requires a little different from general-purpose systems. Firstly, the complexity of memory management system increases because there is more than a single DRAM array. The management of the TLB needs to be more deterministic than it is in typical approach based systems where random replacement was done.

For interface, a DSP-based operating system has to enable several changes on malloc() function, each of which allocates a different virtual region to the process which maps to a separate area of the namespace. An example below shows a set of functions that a DSP based Operating System 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 allots
```

a region within the process address space which is mapped to part of the ROM array. rom_malloc() function requires the software to specify a region in the storage device. This memory-allocation interface should have the required effect of allowing a process to make device-specific decisions of improvement. It protects the virtual address space of processes as well.

4. CONCLUSION

This paper discusses efficient software-oriented cache management schemes for real-time embedded systems by make use of SRAM caches present on-chip. The paper addresses general-purpose caches with DSP caches which share the same namespace similar system used in DRAMs. The address-space protection is provided by a virtual memory technique. Implementation of such a virtual memory layer and address translation adds an extra overhead that can be removed if user-level processes are executed directly on top of memory units.

5. 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 organizations." 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.

