www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 10 October, 2013 Page No. 2944-2950

Cache Memory Organization

Akshay Kanwar, Aditi Khazanchi, Lovenish Saluja

Abstract

The use of cache memories are so persistent in today's computer systems it is difficult to imagine processors without them. Cache memories, along with virtual memories and processor registers form a field of memory hierarchies that rely on the principle of locality of reference. Most applications exhibit temporal and spatial localities among instructions and data. Spatial locality implies that memory locations that are spatially near the currently referenced address will likely be referenced. Temporal locality implies that the currently referenced address will likely be referenced in the near future. Memory hierarchies are intended to keep most likely referenced items in the fastest devices. This results in an effective reduction in access time.

Introduction

Cache memories are used in modern, medium and high-speed CPUs to hold temporarily those portions of the contents of main memory which are currently in use. Since instructions and data in cache memories can usually be referenced in 10 to 25 percent of the time required to access main memory, cache memories permit the executing rate of the machine to be substantially increased. In order to function effectively, cache memories must be carefully designed and implemented. In this paper, we explain the various aspects of cache memoirs and discuss in some detail the design features and trade-offs. A large number of original, trace-driven simulation results are presented.

Cache memories are small, high-speed buffer memories used in modern computer systems to hold temporarily those portions of the contents of main memory which are currently in use. Information located in cache memory may be accessed in much less time than that located in main memory. Thus, a central processing unit (CPU) with a cache memory needs to spend far less time waiting for instructions and operands to be fetched and stored. For example, in typical large, high-speed computers, main memory can be accessed in 300 to 600 nanoseconds; information can be obtained from a cache, on the other hand, in 50 to 100 nanoseconds. Since the performance of such machines is al- ready limited in instruction execution rate by cache memory access time, the absence of any cache memory at all would produce a very substantial decrease in execution speed.

1.1. Cache Memory Principles

- If data sought is not present in cache, a block of memory of fixed size is read into the cache.
- Locality of reference makes it likely that other words in the same block will be accessed soon

2. An Overview of Cache

Cache is small high speed memory usually Static RAM (SRAM) that contains the most recently accessed pieces of main memory. In today's systems, the time it takes to bring an instruction (or piece of data) into the processor is very long when compared to the time to execute the instruction. For example, a typical access time for DRAM is 60ns. A 100 MHz processor can execute most instructions in 1 CLK or 10 ns. Therefore a bottle neck forms at the input to the processor. Cache memory helps by decreasing the time it takes to move information to and from the processor. A typical access time for SRAM is 15 ns. Therefore cache memory allows small portions of main memory to be accessed 3 to 4 times faster than DRAM (main memory).

The theory that explains this performance is called "Locality of Reference." The concept is that at any given time the processor will be accessing memory in a small or localized region of memory. The cache loads this region allowing the processor to access the memory region faster. How well does this work? In a typical application, the internal 16K-byte cache of a Pentium® processor contains over 90% of the addresses requested by the processor. This means that over 90% of the memory accesses occur out of the high speed cache.

2.1. Basic Model

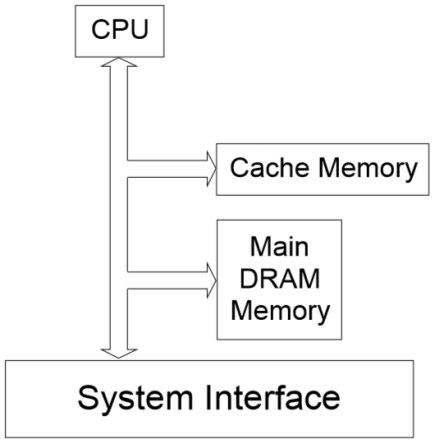


Fig. 1: Basic Cache Model

Fig.1: shows a simplified diagram of a system with cache. In this system, every time the CPU performs a read or write, the cache may intercept the bus transaction, allowing the cache to decrease the response time of the system. Before discussing this cache model, let's define some of the common terms used when talking about cache.

Cache Hits
When the cache contains the information requested, the transaction is said to be a cache hit.

Cache Miss
When the cache does not contain the information requested, the transaction is said to be a cache miss.

Cache Consistency
Since cache is a photo or copy of a small piece main memory, it is important that the cache always reflects what is in main memory. Some common terms used to describe the process of maintaining cache

Consistencies are:

Snoop
When a cache is watching the address lines for transaction, this is called a snoop. This function allows the cache to see if any transactions are accessing memory it contains within itself.

Snarf
When a cache takes the information from the data lines, the cache is said to have snarfed the data. This function allows the cache to be updated and maintain consistency.

Dirty Data
When data is modified within cache but not modified in main memory, the data in the cache is called "dirty data."

Stale Data
When data is modified within main memory but not modified in cache, the data in the cache is called stale data.

2.2. Cache Architecture

Caches have two characteristics, a read architecture and a write policy. The read architecture may be either "Look Aside" or "Look Through." The write policy may be either "Write-Back" or "Write-Through." Both types of read architectures may have either type of write policy, depending on the design. Write policies will be described in more detail in the next section. Let's examine the read architecture now.

2.2.1. Look Aside

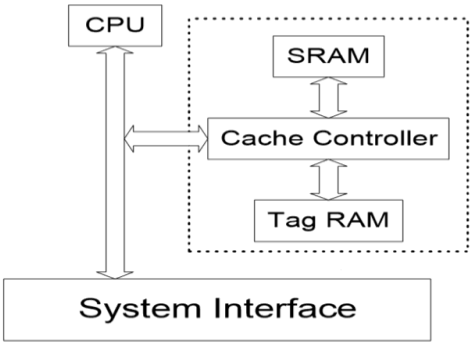


Fig. 2: Look Aside Cache

Fig. 2: shows a simple diagram of the "look aside "cache architecture. In this diagram, main memory is located opposite the system interface. The discerning feature of this cache unit is that it sits in parallel with main memory. It is important to notice that both the main memory and the cache see a bus cycle at the same time. Hence the name "looks aside."

Look Aside Cache Example

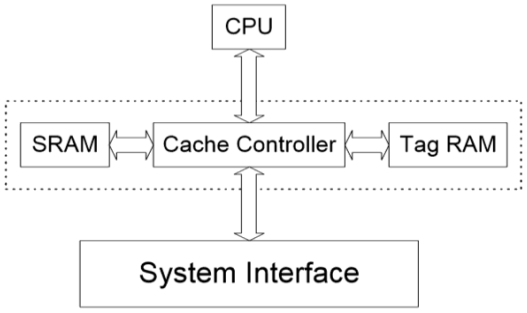
When the processor starts a read cycle, the cache checks to see if that address is a cache hit.

HIT:

If the cache contains the memory location, then the cache will respond to the read cycle and terminate the bus cycle.

MISS:

If the cache does not contain the memory location, then main memory will respond to the processor and terminate the bus cycle. The cache will snarf the data, so next time the processor requests this data it will be a cache hit.



Read Architecture: Look Through

Fig. 3: Look Through Cache

Fig. 3: Shows a simple diagram of cache architecture. Again, main memory is located opposite the system interface. The discerning feature of this cache unit is that it sits between the processor and main memory. It is important to notice that cache sees the processors bus cycle before allowing it to pass on to the system bus.

Look through Read Cycle Example

When the processor starts a memory access, the cache checks to see if that address is a cache hit.

HIT:

the cache responds to the processor's request without starting an access to main memory.

MISS:

The cache passes the bus cycle onto the system bus. Main memory then responds to the processors request. Cache snarfs the data so that next time the processor requests this data, it will be a cache hit.

This architecture allows the processor to run out of

cache while another bus master is accessing main memory, since the processor is isolated from the rest of the system. However, this cache architecture is more complex because it must be able to control accesses to the rest of the system. The increase in complexity increases the cost. Another down side is that memory accesses on cache misses are slower because main memory is not accessed until after the cache is checked. This is not an issue if the cache has a high hit rate and there are other bus masters.

Write Policy

A write policy determines how the cache deals with a write cycle. The two common write policies are Write-Back and Write-Through.

In Write-Back policy, the cache acts like a buffer. That is, when the processor starts a write cycle the cache receives the data and terminates the cycle. The cache then writes the data back to main memory when the system bus is available. This method provides the greatest performance by allowing the processor to continue its tasks while main memory is updated at a later time. However, controlling writes to main memory increase the cache's complexity and cost.

The second method is the Write-Through policy. As the name implies, the processor writes through the cache to main memory. The cache may update its contents, however the write cycle does not end until the data is stored into main memory. This method is less complex and therefore less expensive to implement. The performance with a Write-Through policy is lower since the processor must wait for main memory to accept the data.

2.3. Cache Components

The cache sub-system can be divided into three functional blocks: SRAM, Tag RAM, and the Cache Controller. In actual designs, these blocks may be implemented by multiple chips or all may be combined into a single chip.

2.3.1. SRAM

Static Random Access Memory (SRAM) is the memory block which holds the data. The size of the SRAM determines the size of the cache.

2.3.2. Tag RAM

Tag RAM (TRAM) is a small piece of SRAM that stores the addresses of the data that is stored in the SRAM.

2.3.3. Cache Controller

The cache controller is the brains behind the cache. Its responsibilities include: performing the snoops and snarfs, updating the SRAM and TRAM and implementing the write policy. The cache controller is also responsible for determining if memory request is cacheable and if a request is a cache hit or miss.

2.4. Cache Organization

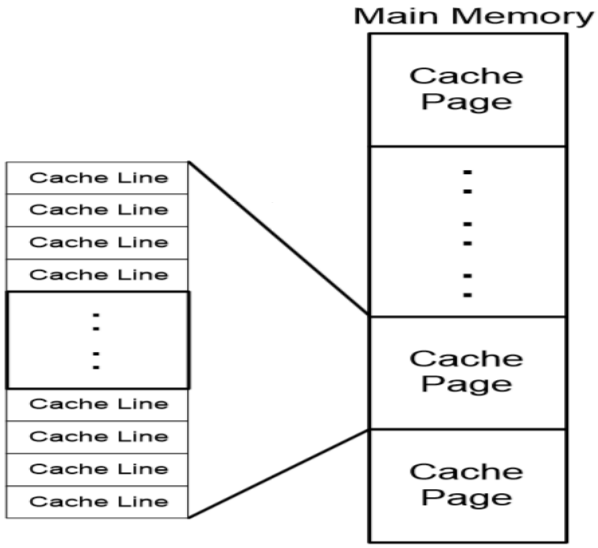


Fig. 4: Cache Page

In order to fully understand how caches can be organized, two terms need to be defined. These terms are cache page and cache line. Let's start by defining a cache page. Main memory is divided into equal pieces called cache pages. The size of a page is dependent on the size of the cache and how the cache is organized. A cache page is broken into smaller piece each.It is not desirable to have all memory cacheable. What regions of main memory determined to be non-cacheable are dependent on the design. For example, in a PC platform the video region of main memory is not cacheable.

A cache page is not associated with a memory page in page mode. The word page has several different meaning when referring to PC architecture called a

cache line. The size of a cache line is determined by both the processor and the cache design. Figure 2-4 shows how main memory can be broken into cache pages and how each cache page is divided into cache lines. We will discuss cache organizations and how to determine the size of a cache page in the following sections.

2.4.1. Fully-Associative

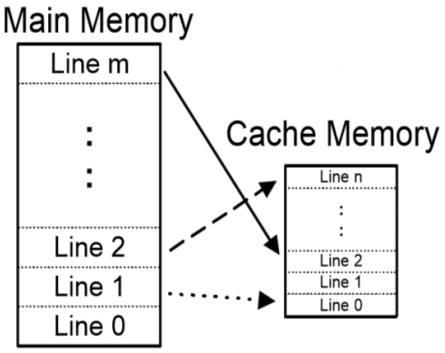


Fig. 5: Fully-Associative Cache

The first cache organization to be discussed is Fully-Associative cache. Fig. 5: shows a diagram of a Fully Associative cache. This organizational scheme allows any line in main memory to be stored at any location in the cache. Fully-Associative cache does not use cache pages, only lines. Main memory and cache memory are both divided into lines of equal size. For example Figure 2-5 shows that Line 1 of main memory is stored in Line 0 of cache. However this is not the only possibility, Line 1 could have been stored anywhere within the cache. Any cache line may store any memory line, hence the name, Fully Associative.

A Fully Associative scheme provides the best performance because any memory location can be stored at any cache location. The disadvantage is the complexity of implementing this scheme. The complexity comes from having to determine if the requested data is present in cache. In order to meet the timing requirements, the current address must be compared with all the addresses present in the TRAM. This requires a very large number of comparators that increase the complexity and cost of implementing large caches. Therefore, this type of cache is usually only used for small caches, typically less than 4K.

2.4.2. Direct Map

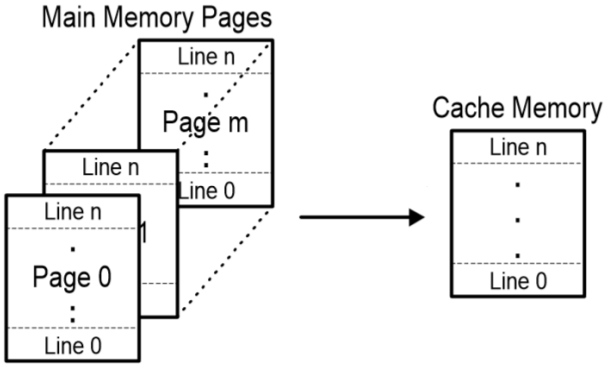


Fig. 6: Direct Mapped

Direct Mapped cache is also referred to as 1-Way set associative cache. Fig. 6: shows a diagram of a direct map scheme. In this scheme, main memory is divided into cache pages. The size of each page is equal to the size of the cache. Unlike the fully associative cache, the direct map cache may only store a specific line of memory within the same line of cache. For example, Line 0 of any page in memory must be stored in Line 0 of cache memory. Therefore if Line 0 of Page 0 is stored within the cache and Line 0 of page 1 is requested, then Line 0 of Page 0 will be replaced with Line 0 of Page 1. This scheme directly maps a memory line into an equivalent cache line, hence the name Direct Mapped cache.

A Direct Mapped cache scheme is the least complex of all three caching schemes. Direct Mapped cache only requires that the current requested address be compared with only one cache address. Since this implementation is less complex, it is far less expensive than the other caching schemes. The disadvantage is that Direct Mapped cache is far less flexible making the performance much lower, especially when jumping between cache pages.

2.4.3. Set Associative

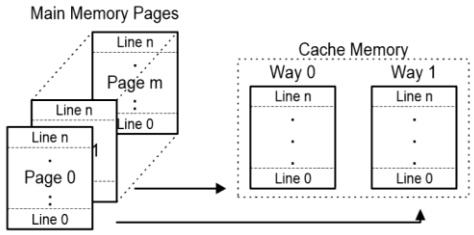


Fig. 7: Way Set Associative

A Set-Associative cache scheme is a combination of Fully-Associative and Direct Mapped caching schemes. A set-associate scheme works by dividing the cache SRAM into equal sections (2 or 4 sections typically) called cache ways. The cache page size is equal to the size of the cache way. Each cache way is treated like a small direct mapped cache. To make the explanation clearer, lets look at a specific example. Fig. 7: shows a diagram of a 2-Way Set-Associate cache scheme. In this scheme, two lines of memory may be stored at any time. This helps to reduce the number of times the cache line data is written-over.

This scheme is less complex than a Fully-Associative cache because the number of comparators is equal to the number of cache ways. A 2-Way Set-Associate cache only requires two comparators making this scheme less expensive than a fully-associative scheme.

3. Types of Cache Memory

	Memory Types	Description
1	**Soft reference cache**	When objects are removed from the memory cache in order to keep the memory cache size constant, they are moved to a soft reference cache, which can grow or shrink based on the available memory. If the JVM has to reclaim memory space, it takes it from the soft reference cache.
2	**Memory cache**	Uses the amount of memory necessary to hold the objects in memory at all times. You can configure this in the noapp.properties file.
3	**Disk cache**	Objects can be read more quickly from the disk than from the database. When objects are no longer in the soft reference cache because they have been garbage collected, the disk cache provides a faster access mechanism than object retrieval from the database. You can configure this in the noapp.properties file.

4. Features of cache

• The access time is faster than main memory.

• The cache is placed on the top in the memory hierarchy.

• The cache organization is concerned with memory write requests.

• The cache is initialized when the main memory is loaded with a complete set of programs.

• Cache built into the CPU itself is referred to as Level 1 cache.

• Cache that resides on a separate chip is called Level 2 Eg: SRAM

5. Advantages and Disadvantages of Cache

Advantages

• It is the fastest memory in the system.

• Cache is so effective in system performance

• The CPU does not have to use the motherboard's system bus for data transfer.

• The CPU can process data much faster by avoiding the bottleneck created by the system bus.

• Programs can operate more quickly and efficiently.

Disadvantage

• Cache is more expensive than RAM.

6. Conclusion

Cache memories play a important role in recuperating the performance of today's computer systems. Copious techniques for the use of caches and for maintaining coherent data have been proposed and implemented in commercial SMP systems. The use of cache memories in a distributed processing system, however, must be carefully implicit to benefit from them. In a related research on memory consistency models, recital improvements are achieved by requiring data consistency only at synchronization points. The research on cache memories in distributed systems is very active. Furthermore research is currently active on the use of cache memories in fully distributed systems, including web-based computing.

9.

References

1. **technav.ieee.org/tag/4457/cache-memory**
2. **en.wikipedia.org/wiki/CPU_cache**
3. **en.wikipedia.org/wiki/Cache**
4. **http://www.webopedia.com/TERM/C/cache.html**
5. **http://www.wisegeek.org/what-is-cache-memory.htm**
6. **http://www.computerhope.com/jargon/c/cache.htm**
7. **pic.dhe.ibm.com/infocenter/sb2bi/.../SIPM_TS_CacheMemTypes.html**
8. **personal.cityu.edu.hk/~dcykcho/dco2230/chappp2/sld028.htm**