Cache: Why Level It

Nuno Miguel Duarte Cerqueira Dinis

Departamento de Informática, Universidade do Minho
4710 - 057 Braga, Portugal
nunods@ipb.pt

Abstract. Processing speed is faster every day; however all other components are staying behind in this run for speed, particularly access times to the memory hierarchy. Cache memories play an important role in this gap. This communication gives some clues how to improve cache performance, and debates the alternative of larger versus layered caches, based on spatial locality, latency, cost, miss rate and efficiency.

1 Introduction

Let us revise in a simplistic way the concept of the whole memory subsystem, known as a memory hierarchy. This means that most of the computer systems have multiple levels of memory, each level commonly being of a different size, and different speed. The fastest is the cache and it is the one that is closest to the processor (in modern ones, inside the processors chip), and each subsequent layer gets slower, farther from the processor, and (generally), larger until the big storage devices (HD, DAT, etc). One of the reasons why each progressively lower level on the memory hierarchy must be larger is due to the fact that each layer keeps a copy of the information that is in the smaller/faster layer above it. What this means is that the hard drive holds the information that is in the RAM, which holds information that is in the cache.

1.1 Why Cache

Following the computer evolution, every processor itself is starving for bandwidth, since to keep their good performance it needs memories to support these processing levels. If it were not for the cache memory with an incredible bandwidth, all the performance would become ridiculous. Just because the data acquisition from memory to be processed would drastically damage the efficiency of the processor.

So it is not an accident that all processors have currently cache memory, and most of them several levels of it, as it will be later mentioned in this communication.

However, there are times when this supposed theory does not apply. When the computer is manipulating, and creating large amounts of data, and not just reading stuff from the computer. An example of this would be for large amount of scientific calculations, where the processor does not need to write the disk very often, and because of that the data stored in the cache and main memory is not accessed very often.

The term “locality reference” works like a rule in computer science/computer architecture that is the following: ”Programs tend to reuse data and instructions they have used recently. A widely held rule of thumb is that a program spends (about) 90% of its execution time in only (about) 10% of the code”. If a processor uses only 10% of the code most of the time, then why not keep that code close to the processor so that it can be accessed faster?

The structure of a cache memory is different from the main memory, as shown in Fig 1 below:

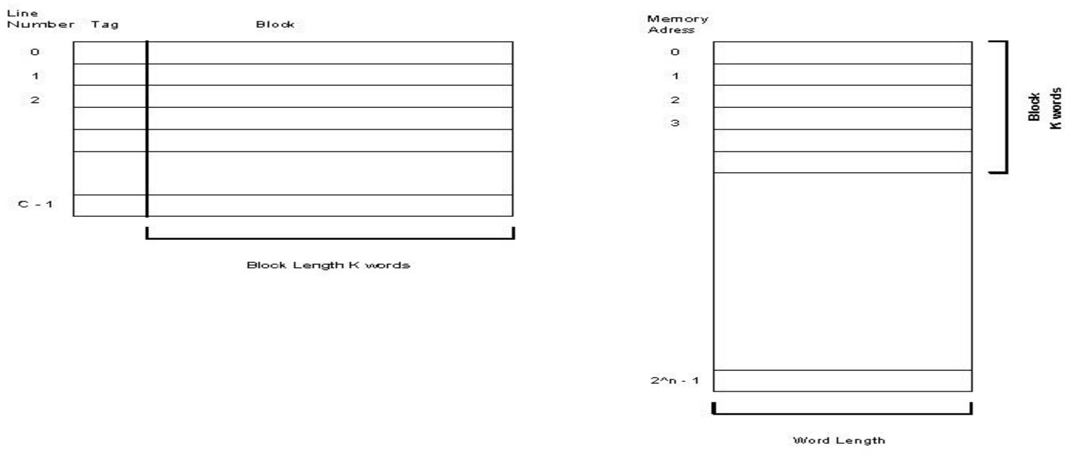

Fig 1. Cache/Main-Memory structure [2]

When the processor needs a word, it generates a RA (Reference Address), if the word is in cache block, then it is delivered to the processor, if not the block in the main memory containing the word is loaded to the cache and the word to the processor. Fig. 2 shows a diagram representing the reading operation using cache memory.

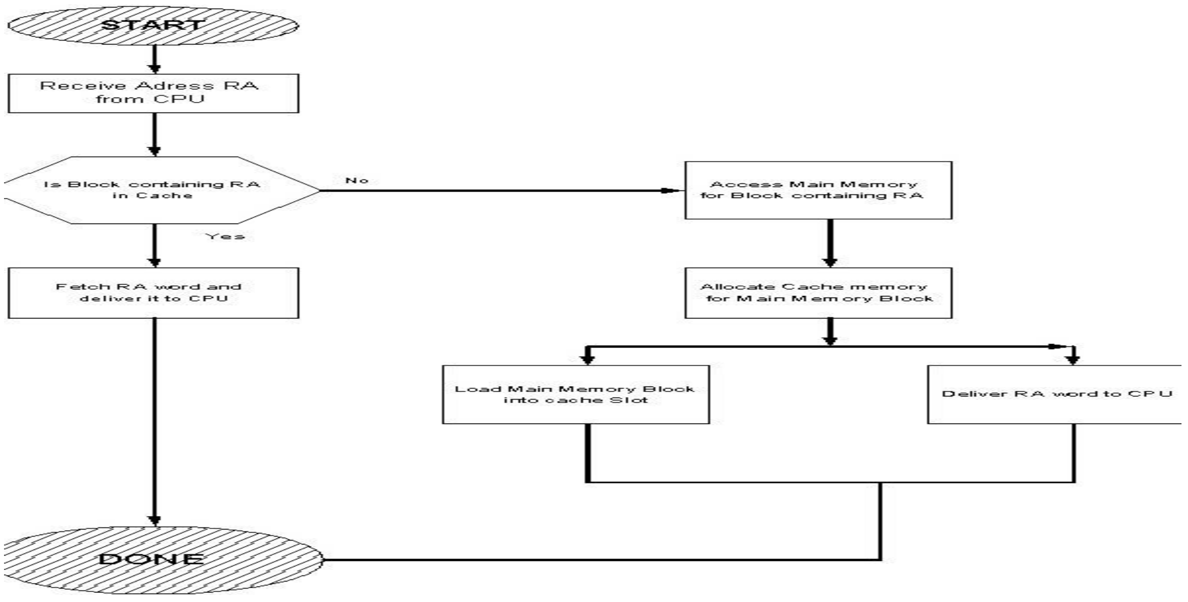
START
Receive Adress RA from CPU
Is Block containing RA in Cache
No
Yes
Access Main Memory for Block containing RA
Fetch RA word and deliver it to CPU
Allocate Cache memory for Main Memory Block
Load Main Memory Block into cache Slot
Deliver RA word to CPU
DONE

Fig 2. Cache read operation [2]

1.2 Cache Issues

Cache-line is the amount of data transferred between the main memory and the cache by a cache-line fill or write-back operation. So its size is a very important issue. The larger the cache-line size, the more data that is close together, and therefore, likely related, is brought into the cache at one time.

Latency is the time it takes a task to be executed; this characteristic is expressed in clock cycles from the perspective of the device's clock. For example, 100mhz SDRAM with a latency of 9 cycles with a 1ghz CPU means a latency of 90 cycles to the processor! In the case of cache and memory, it refers to the amount of time that it takes for the cache (or memory) to send data.

Cache hit refers to a situation when the processor asks for data from the cache, and gets it.

Cache miss is a situation when the processors asks for data from the cache, and does not get it.

Hit rate refers to the average percentage of times that the processor will get a cache hit without having a miss.

1.3 How Cache Works

Cache memories have several different working methods. These methods are associated with their design, since it directly implicates with the way data flows between the processor and the cache. So here are the most common cache designs:

Direct Mapped Cache: Each memory location is mapped to a single cache line that it shares with many others; only one of the many addresses that share this line can use it at a given time. This is the simplest technique both in concept and in implementation. Using this cache means the circuitry to check for hits is fast and easy to design, but the hit ratio is relatively poor compared to the other designs because of its inflexibility. Motherboard-based system caches are typically direct mapped.

Fully Associative Cache: Any memory location can be cached in any cache line. This is the most complex technique and requires sophisticated search algorithms when checking for a hit. It can lead to the whole cache being slowed down because of this, but it offers the best theoretical hit ratio since there are so many options for caching any memory address.

N-Way Set Associative Cache: "N" is typically 2, 4, 8 etc. A compromise between the two previous design, the cache is broken into sets of "N" lines each, and any memory address can be cached in any of those "N" lines. This improves hit ratios over the direct mapped cache, but without incurring a severe search penalty (since "N" is kept small). The 2-way or 4-way set associative cache is common in processor level 1 caches.

1.4 Cache Performance

Talking about figures, they only can tell if cache is really an improvement to the performance. The highest performance of a CPU is achieved when the CPI (cycles per instruction) is equal to 1, in a scalar architecture.

For a given example, without any cache memory, the CPU would have a CPI of 5.14; by duplicating the processors speed the CPI would become 4.775 that would not be much of an improvement. If the main memory would become faster by 30%, then the CPI would be 4.01, a significant improvement. But if the hit rate was increased by 10%, then the CPI would be even lower: 2.71.

2 How to Get More Effective Caches

These are some parameters that can improve the performance of the cache memories:

Cache block size (larger cache lines) - The larger the cache-line size, the more data that is close together, and therefore, likely related is brought into the cache at any one time. The CPU only requests a small piece of information, but it will get whatever other information is contained within the cache-line. If the cache is large enough, then it can easily contain the information within a large cache-line. However, if the cache is too small in comparison to the cache-line size, it can reduce performance (because sometimes, irrelevant information is in the cache-line, and takes up valuable space).

More placement choice (more associativity) **-** Given the same latencies, the more associative a cache is, generally the higher the hit rate and better performance received. However, the larger the cache, the more difficult it is to get it to reach both high clock speeds and low latencies. Clock speed is very important in terms of bandwidth, and latencies and bandwidth go hand in hand.

Innovative caches - these caches mess with the nature of the cache exclusivity. The relationship is only a L1/L2 relationship. What this means is that, it is solely exclusive between the L1 and L2, and that the information in the L1 is duplicated in the main memory (and, potentially, though not always or often, in the hard drive), yet not in the L2 cache.

However this communication aims to talk about a *cache hierarchy* versus *larger caches*, since this is probably the biggest discussion point after getting to the conclusion that hit rate time was a big and important factor to cache performance.

3 Larger versus Layered

3.1 Numbers between Big and Small

Table 1 presents some values of 3 types of cache memories that may not correspond to current sizes, but they can give an idea on the implications that changing the size has on the hit rate (bigger size better hit rate) and access time (bigger size worst access time).

Size	Hit rate	Miss rate	Access time
4KB-8KB	90%	10%	1ns
64KB-256KB	99%	1%	5ns
256KB-2MB	99.8%	0.2%	10ns

Tab. 1 Cache memories characteristics

3.2 Advantages and Disadvantages of Larger Caches

Increasing a single cache module, may have several consequences, some good others not that good. Here we will be talking about them.

Cache is a kind of memory very expensive, and it is usually made of SRAM (static RAM), and it is different from the regular main memory designated by EDO, SDRAM, etc. The cache memory uses a lot more transistors for a single bit of information, and because of this two issues are consequently changed, one is that more components more space they use inside the processors chip, another is that electrical consumption of energy increases, and if we start thinking that electrical consumption is an important factor when the philosophy of the computers in our days is smaller and portable and to achieve that with this characteristics it can be very difficult.

Otherwise enlarging the cache memory would have several improvements to the performance. If it is large enough, then it can easily contain the information within a large cache-line. Another important factor is that in larger cache memories the hit rate increases (ideal hit rate 100%). But increasing the hit rate is a determinant factor in the access time, the better hit rate the worst access time, and to get this high performance in the hit rate the efficiency in terms of access time may become similar to main memory.

A small better/worst table (Table 2) can be presented showing the analysis factor for a larger cache memory:

Better/Worst	Factors
Better	The safest way to get improved hit rate
Worst	SRAMs are very expensive
Worst	Larger size => lower speed
Worst	Power consumption
Worst	Reliability

Tab. 2 Larger memory factors

3.3 Multi Level Caching

Let us create a simplistic example to see if it can be useful understanding the concept of multi-level caching. If you work in a big warehouse every time a client comes to you asking for a product you take a lot of time to find it, and if the product is at the end of the warehouse it would took even longer to get the product. Suppose you attend 200 clients a day and 75% of the products they ask you for are always the same (for example 400 different products). If a small room is built next to the costumer's reception point and if you put in that room those products that are used more often maybe the client would be satisfied. However from those 400, 100 are asked 50 % of the times, so why not create another room a little bit bigger to store 300 different products, and in that one 100 (Fig. 3).

So it seems like it is a good solution for achieving a product in a short amount of time, let us try to import this simplistic example to the cache hierarchy reality.

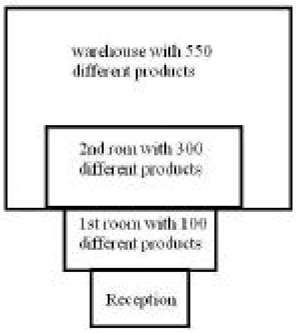
warehouse with 550 different products
2nd room with 300 different products
1st room with 100 different products
Reception

Fig. 3 Example of a 3-stage warehouse

3.4 Technical Factors

First of all a main aspect to start thinking on layering the cache memory was the possibility that modern processors offered to have inside the chip a module of cache memory. This was a big advantage since it would eliminate bus delays, and the access time would be almost zero.
But then was it necessary to have other levels of cache?

The answer is yes. Because if the desired data was not found on the internal cache then it would have to be searched in the main memory (very slow), then why not insert another level of cache in the middle of the inside cache and the main memory.
After the 2 levels of cache started being implemented another idea appear that was to split the inside cache into two caches one for data and another for instructions. However this has some problems. It is necessary to design and implement two caches, and for a unified cache the hit rate is higher because it automatically balances the load between instruction and data fetches. Even so the evolution is tending to use split caches.

The benefits of cache hierarchy come at a price. Caches higher in the hierarchy must field the misses of their descendents. If the equilibrium hit rate of a leaf cache is 50%, this means that half of all leaf references get resolved through a second level cache rather than directly from the object's source. If the reference hits the higher level cache, so much the better, as long as the second and third level caches do not become a performance bottleneck. If the higher level caches become overloaded, then they could actually increase access latency, rather than reduce it.

The evolution of several cache layers started to be a reality, it is good for performance. Despite the cache speed difference between L1 and system memory still meant every time the CPU needed to access system memory, it had to request the data, and then wait for it. A small L1 cache was nice, but it wasn't enough.

A level-2 (L2) cache is usually fast (25ns or faster) RAM. Cache size can range from 8KB to 1 MB. There is no hard and fast rule about cache size, but the general rule is the faster the CPU relative to system memory, the larger your L2 cache should be. And then why not have a third level of cache (L3) with a bigger size then L2. Next is a diagram (Fig. 4) showing a architecture with 2 levels of cache inside the processor (being the L1 spitted) and L3 in the CPU board:

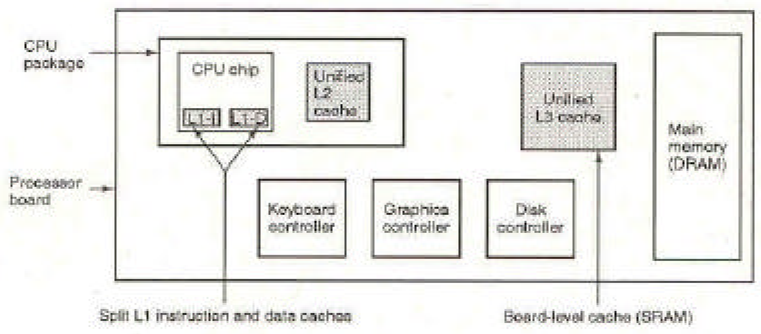

Fig. 4 Layout of a 3 level cache memory

Let us create a small simulation on the improvement numbers that increasing the cache levels can do.

50 ns Main memory
L1, 1ns, 4KB, (10% miss rate)
L2, 5ns, 128KB, (1% miss rate)
L3, 10 ns 1MB, (0.2% miss rate)

- No cache
  CPI=base+0.3*50=base+15
- One level of cache
  CPI=base+0.3*0.1*50=base+1.5

- Two levels of cache
  CPI=base + (0.3*0.1*5+0.01*50)=base+0.65
- Three levels of cache
  CPI=base + (0.3*0.1*5+0.01*10+0.0002*50=base+0.35

4 Conclusions

After researching and the making of this communication, I think that nothing can be said as being the best solution. But from the point of view of almost all manufactures the principle of creating several levels of cache memory is the way to the solution instead of thinking of a single and big memory, however the discussion appears when discussing several kinds of architectures and sizes between L1, L2 and L3. Since almost each builder has a different architecture for the data flow and storage between cache levels, and they also have different perspectives because each builder uses different kinds of cache memories with different performances.

As for the future, it is likely that we will not be seeing larger and larger L1 caches on CPUs (with the exception being HP PA-RISC processors), because of the ability to economically integrate L2 cache on-die. However a processor builder had a discussion saying that L1 cache could become smaller for best performance. Their reasoning was that large caches would not allow for low latencies while maintaining high clock speeds, which is what the processor needs from an L1 cache, and the on-die L2 cache would give the high hit rates.

Some other competitors are following this concept. These builders have high bandwidth L2 caches on-die. However all the evolution of L1 size is developing against this theory.

So from my point of view (a simple master's student) I think that the economical factor is interfering with losing out technology to the common market. For example a builder starts developing a super machine (spending time and money) with very high performance in what concerns to cache memories, but would the turn over compensate, since this machine would become very expensive (more cache memory more expensive the computer system). So I think that almost all of them are trying to get the best performance without having to use a big amount of cache memory.

The market prices laws interferes almost every evolution in our days: evolution yes, sell a lot yes, but to sell a lot it as to be at a cheaper price.

References

[1] Hwang, Kay, Xu, Zhiwei: Scalable Parallel Computing, McGraw-Hill, 1998

[2] Stallings, William: Computer organization and Architecture, 5th edition, 1999

[3] Hennessy, Patterson: Computer Architecture: A Quantitative Approach

[4] "AMD Athlon ™ Processor and AMD Duron ™ Processor with Full-Speed On-Die L2 Cache Enabling an Innovative Cache Architecture for Personal Computing."

[5] DeHon, Andre: Computer Architecture (Single Threaded Architecture: abstractions, quantification, and optimisations), Caltech CS184b 2001

[6] Gelas, Joan De: Ace’s Guide to Memory Technology, Ace’s Hardware http://www. aceshardware.com/Spades/read.php?article_id=5000173

[7] Gelas, De Joan: A preview of the fastest PC processors in the year 2000, Ace’s Hardware, http://www.aceshardware.com/Spades/read.php?article_id=86

[8] Yeap, KH and JC: Rise Technology mp6 preview, http://www.jc-news.com/pc/ article.cgi?Rise/mp6_preview