Stephanie Honore
Dumas
CPSC 5700: Computer Architecture
January 26, 2017

Computer Memory Systems:A Summary of Cache Write Policies, Cache Replacement Strategies, and, Cache Initialization

By Stephanie Honore`

The purpose of this paper is to give a summarized description of the following topics: cache write policies, cache replacement strategies, and, cache initialization. Before these subjects can be discussed in further detail, it is important to give a brief background of what cache memory is and how it designed from an architectural standpoint.

Cache memory is a smaller and faster form of intermediate memory that typically exists between the CPU and the main memory. The main memory is DRAM, or dynamic random access memory, while CPU cache is a form of SRAM, or static random access memory. In the memory hierarchy of a typical modern-day computer systems, this type of memory would be ranked relatively high compared to other types of memory systems due to it's fast availability of access. The reasoning behind these being the fastest form of semiconductor read/write memory technology is because they store the states of the binary information in flip-flops instead of utilizing charged capacitors like DRAM. The main catch with implementing cache memory in a computer system is the high monetary cost.

Stephanie Honore

Dumas

CPSC 5700: Computer Architecture

January 26, 2017

Cache memory ideally would hold the data and information that needs to be accessed quite frequently. When storing information in the cache versus main memory ,the computer and most importantly the end-user would be able to benefit from the perception that the memory system is faster than it truly is. The concept of cache memory is possible due to the principle of locality. The principle of locality (or locality of reference) “states that programs tend to access code and data that have recently been accessed, or which are near code or data that have recently been accessed” (Dumas 77). There exists 3 main elements behind the principle of locality, which are temporal locality, spatial locality, and, sequential locality. All 3 of theses concepts involve how memory locations are accessed and cache memory takes advantage of all 3 of these forms of locality to function as we know it.

Despite what the name implies, access to RAM is not really random. In fact, an access pattern is established by identifying areas that are easier to access and by using the same items over and over again. In addition, items that are in close proximity to previously accessed items are used too. Essentially, the patterns of access that cache memory uses are concentrated and localized. When the information that is requested is located in the cache memory then a cache hit occurs. When the information is not located in the cache memory, but is located in the main memory or any of the lower levels of memory then a miss occurs. The performance of cache is measured through the hit-ratio. The hit-ratio is defined as the number of hits over the total number of memory accesses. The total number of memory accesses is the combined number of

Dumas
CPSC 5700: Computer Architecture
January 26, 2017

hits and misses. The hit ratio is measured between 0 and 1 and between 1 and 100 percent. The overall access time depends on the hit ratio and the miss ratio. The hit ratio is not a constant value on a single machine, with it varying depending on which programs are requesting access to main memory. This is because the longer you run a program, the more of its data is stored in cache memory. Basically, the longer the program is running the faster it will become.

When a computer architect is designing the memory system of a computer they need to decide what goes where in a cache memory system. This happens on a demand basis and through a mapping strategy. When one says a demand basis in reference to computer architecture, this means that they want to make sure that the items that are cached (or in demand) are the ones being actually used or are local to the items in use. Items are not cached individually, but are cached in blocks. In addition, items are cached according to a refill line, which is a unit of information that is transferred between 2 levels of memory (in this case the cache memory and the main memory).

Next, it is important to cover memory mapping strategies. This orchestrates where everything goes in a cache. There are 3 main types of mapping strategies highlighted in the text when it comes to cache memory systems. These are the fully-associative cache, direct mapping cache, and the n-way set-associative cache. In most modern computing systems the cache memory system is located on a integrated chip with the CPU and registers and it contains a replica of a commonly used blocks of main

Dumas

CPSC 5700: Computer Architecture

January 26, 2017

memory, where one line in the cache contains a block of main memory. Something called a tag identifies which block of main memory is stored in that line of cache. A block in main memory contains words, which is a collection of bits, and an associated memory address.

The direct mapping technique is the least complex, yet least flexible way of mapping. This involves each block of main memory mapping to a unique line of cache memory. The more complex, yet more flexible way of mapping technique is fully associative mapping. This technique allows each main memory block to be loaded into any line of cache. To determine whether or not a block is located in the cache memory, cache control logic examines every line's tag value for a matching address value in main memory simultaneously. The middle ground between direct mapping and associative mapping is set-associative n-way mapping. With this technique, cache is split into n sets. A block of main memory can be mapped into any of the lines within the associated set in cache that the block is assigned to.

Now, after having a general overview of what cache is and how it is designed we can now move on to cache write policies. According to Dumas, writes make the cache design process complex since you must take into consideration "what must happen when a line is displaced from cache" (Dumas 87). What makes writing to cache so complex is coordinating with main memory to make sure the information is up-to-date with the current contents. In the text, two cache write policies are discussed in detail, these are write-through cache and write-back cache.

Stephanie Honore

Dumas

CPSC 5700: Computer Architecture

January 26, 2017

Write-through cache is the process of writing to both the cache location and the matching main memory location simultaneously. This type of write policy has it's advantages and disadvantages. The plus side is that implementing a write-through cache is relatively simple and the information in the main memory can stay current and consistent with the corresponding cache memory. The downside is that accessing and writing to the main memory location takes more time than writing just to the cache memory location so, the overall process takes a longer time. Another disadvantage is that the performance of the machine could be inhibited since the cache hit ratio might end up being lower. The memory system is designed so that read hits are beneficial in terms of performance while write hits actually behaves like a cache miss. This is due to an item being accessed multiple times while it is in cache. Imagine the principle of locality, particularly temporal locality. If a line in the cache is accessed to be read or written to, then a short span of time later it is likely to be reaccessed to be read or written to again. This would entail the same line of main memory being accessed to be read or rewritten to again as well (Rivoire "Cache Write Policies").

The other type of write policy mentioned is write-back cache. This type of cache write policy only updates the cache memory when a cache hit occurs and only writes to main memory when a line in the cache is displaced for new data. Compared to write-through cache, the performance would be ranked higher because writes to main memory would happen very infrequently. The trade-off of using write-back cache is that it is more complex to implement than write-through cache. When designing the

Dumas
CPSC 5700: Computer Architecture
January 26, 2017

architecture of this system something called a “dirty bit” must be included with the tag. The “dirty bit” would be a 1 if they associated line has been read and written to already in the cache and it would be a 0 if the line has not been written to cache already, but only read. After reading and writing to the cache the value of the dirty bit would change from 0 to 1. If the value of the dirty bit is already 1 when the read occurs and the contents of the cache block must be overwritten for new information to be added, then the information in the block will be written back to the main memory.

Unlike the write-through cache technique, write hits are not detrimental to performance, in fact they are advantageous. This is mainly due to the fact that a line is only written to main memory once versus multiple times. The downside of using this cache write policy is that the information in main memory and subsequently secondary memory devices can be out-of-date and inconsistent for much longer periods of time. Also, when implementing this type of cache write policy a buffer must be created to stored the displaced information until it is ready to be written to main memory. This requires a more complex form of logic than the write-through cache.

In a scenario where the cache is full and new information needs to be written to cache, information that is no longer needed or information that is irrelevant must be displaced from the cache. This process does not happen at random, but in a actuality a value function is applied to each of the cached items and the item with the lowest value is chosen to be replaced. In order to do this a proper cache replacement strategy must be implements so that the memory system maintains it’s highest performance. In most

Stephanie Honore

Dumas

CPSC 5700: Computer Architecture

January 26, 2017

modern systems the mentioned value function is based on the principle of locality , particularly temporal locality, semantic locality, and spatial locality. Semantic locality is similar to spatial locality, however it dynamically adapts the location pattern of the referenced data(Dar, Franklin, Jonsson, Srivastava, Tan 332). Keeping this in mind, the type of replacement strategies that exist today do not vary much in terms of performance so simplicity and versatility are the major aims of the write replacement strategies. This involves the construction of a simplistic, yet effective algorithm that can seamlessly decide where the new line should be loaded into the cache memory. When concerning direct mapping cache this problem is easy to solve “since there is only one place a given item can go” (Dumas 89). In a situation where fully associative mapping or set associative mapping occurs then this process can get tricky.

One type of algorithm that could be utilized to decide which line of data to displace is the Least Frequently Used algorithm. Other types mentioned are the Least Recently Used, First In First Out, Round Robin, and Random algorithms. The Least Frequently used, or LFU, algorithm chooses the line or lines in memory that are used the least. This is determined by the hit-ratio. So lines that have the lowest hit-ratio value are the ones that are overwritten first. For Least Recently Used(LRU for short), the temporal locality of the line is assessed. The lines in memory that were accessed the longest amount of time ago are the ones that are chosen to be displaced. First In First Out (FIFO) functions like a queue data structure where lines are displaced in chronological order (). So, the lines that were written first are the ones to be overwritten

Dumas

CPSC 5700: Computer Architecture

January 26, 2017

first. The Round Robin algorithm involves the cache lines rotating and taking turns being replaced without in discrimination as to the contents of the line. A random algorithm can be constructed to decide which line to displace as well. Although this is not ideal, it is acceptable since, as mentioned earlier, the write replacement strategy does not have a huge impact on performance in the long term.

The final major topic of this paper is cache initialization. When a machine resets or starts a new process the cache is initially filled with either residual data from a previous process or something called, “garbage data”, which is a random assortment of 1’s and 0’s. Regardless, the cache is always filled with some sort of data. To overcome this dilemma a valid bit is assigned to the tag of the memory line. The purpose of the valid bit is to make sure that after a reset or the beginning of a new process the cache does not read the instructions from the uninitialized garbage data to the CPU. The valid bit functions similar to the dirty bit in that it is associated with the cache memory’s tag. The process of initializing the cache involves setting a valid bit to the correct value after receiving cache misses for all requested cache accesses and then retrieving the line of data from main memory. After this the valid bit is considered set and the associated tag can be accessed to read and write data in the cache memory. A valid tag hit has a value of 1 when a true hit occurs. It is possible to lock entries into place in cache so that when the system restart the cache is filled with as much initialized and valid data as possible. This would improve the overall performance of the computer because it would decrease the amount of time it would take to locate the lines of data in main memory. This

Stephanie Honore

Dumas

CPSC 5700: Computer Architecture

January 26, 2017

essentially leads to a better hit-ratio over a shorter span of time than if the valid information was completely flushed from the cache on reset.

Research sponsored by the IEEE Computer Society found that a process called temporal data prefetching reduced the number of cache misses in servers workloads. The specific temporal prefetching technique is known as Domino prefetching. This technique looks "looks up the history to find the last occurrence of the last one or two L1-D miss addresses for prefetching" (Bakhshalipour, Lotfi-Kamran, Sarbazi-Azad 1). This relates to cache initialization because when a cache is flushed on hardware reset or under supervisor software control initially all of the accesses to the cache are forced to be misses. By using this Domino prefetching you could reduce the time it would take to access main memory and see significant improvement in performance. According to this research, system performance on a server workload increased by 26% on average and 56% in the best case scenrio.

Stephanie Honore

Dumas

CPSC 5700: Computer Architecture

January 26, 2017

Works Cited

1. Bakhshalipour, M., P. Lotfi-Kamran, and H. Sarbazi-Azad. "An Efficient Temporal Data Prefetcher for L1 Caches - IEEE Xplore Document." IEEE Computer Architecture Letters 99 (2017): 1. An Efficient Temporal Data Prefetcher for L1 Caches - IEEE Xplore Document. IEEE Computer Society, Jan. 2017. Web. 01 Feb. 2017
2. Dar, Shaul, Michael Franklin, Bjorn Jonsson, Divesh Srivastava, and Michael Tan. "Semantic Data Caching and Replacement." Semantic Data Caching and Replacement (1996): 332. VLDB Endowment. Endowment Inc., 1996. Web. 1 Feb. 2017.
3. Dumas, Joseph D. Computer Architecture: Fundamentals and Principles of Computer Design. 2nd ed. Boca Raton, FL: CRC, 2017. Print.
4. Rivoire, Suzanne. "Cache Write Policies." CS 351 Course Information. Sonoma State University, Jan. 2017. Web. 1 Feb. 2017