Cache Research Paper Kyle Drewes

For this research paper, after viewing the presentation orchestrated by Jason Low-Power, I decided to conduct my own research on the study of Caches.  Specifically, my research will include what a cache is, why it is used, how it operates and its overall advantages and disadvantages.  In this research paper, I will discuss what a cache is, how it operates and its main purpose of utilization.  From there, I will discuss the technicalities involved in caching such as the different levels used, cache placement policies and the overall advantages and disadvantages of using a Cache.

The first question someone may have is “what is a Cache?”  The Cache is a component of the CPU which used for the purpose of storing memory that has been originally comprised from the computer’s physical/main memory or RAM.  The purpose of utilizing a cache is to store data that is used more commonly, and most recently by the computer’s CPU.  By storing data within the Cache, the CPU will have faster access time which allows the computer to preform more effectively.  After all, the more accessible the data is, the faster the computer performs.  The less accessible the data is, the slower computer performs.  If the CPU is instructed to fetch a particular instruction that is not found in the cache, then it will be forced to search for the instruction within the computer’s RAM.  As a result, this generates more latency and decreases the computer’s overall performance.

The way a cache works is when the CPU attempts to access data, it will examine the cache to see if the data is currently present.  If the data is present, then this will cause a hit.  As a result, the CPU will be able to successfully retrieve the necessary data.  However, if the data is not found, then this will cause a miss to occur.  Once a miss occurs, the CPU will have to go back to memory to retrieve the data, then insert the data back into the cache.  From there, the cache will be able to access the data in the future.  Caches have the ability to retrieve data through two different types of locality known as *spatial locality and temporal locality.*  *Temporal Locality* is the idea that if a particular memory address was already previously retrieved by the cache, then that memory address will most likely be retrieved again.  On the other hand, *spatial locality* is the concept that if a memory address has been retrieved by the CPU and is currently stored in the cache, then any memory address that is close to the memory address will most likely be retrieved by the CPU as well.

In contemporary CPU’s, there typically are three different levels of cache: L1, L2 and L3.  L1 cache is undeniably the fastest level of cache which normally holds anywhere from 256 KB to 1 MB of data.  Not only is L1 the fastest and easiest level of cache to access, but it contains two separate components: L1 Data Cache and L1 Instruction Cache.  The L1 Instruction Cache is used for the traditional purpose of a Caches, which is to store data that the CPU will subsequently use to perform a particular operation.  However, The L1 Data Cache contains information is used to transfer back into memory when request by physical memory.  In other words, the L1 Instruction Cache is read by the CPU while the L1 Data Cache is written back to memory.    When the L1 cache is fully occupied, then the CPU will use the L2 cache as a backup source.  The L2 cache is larger than the L1 cache, consisting of a size of 1MB to 10MB.  However, it is not as fast as the L1 cache because it is more difficult to access.  However, it is much more efficient to utilize than the physical memory.  Lastly, L3 cache is utilized as the final backup cache is L1 or L2 is unable to be access by the CPU.  The L3 cache contains approximately  16MB to 64MB.  However, it is considered as the slowest cache on the CPU and should only be used as a last resort.  If L1, L2 or L3 is unable to be accessed, then the CPU will be forced to search for the necessary data within the RAM.  Another difference between L1, L2 and L3 cache is each core processor consists of its own set of L1 and L2 cache.  On the other hand, the L3 cache is shared between each core processor.

When utilizing caches, each cache is going follow a *cache placement policy*.  A cache placement policy is a type of method used by the cache to store data within a cache block.  Depending on the cache placement policy, each cache may have a different technique for storing memory.  You can think of a Cache Placement Policy as a technique used to distribute the cache blocks within the cache.  Depending on the cache placement policy, the data within the cache blocks will be organized differently.  In this next part, I will be discussing the layout of how a cache is generally constructed.  From there, I will discuss the three different types of cache placement policies: Direct Mapped Cache, Fully Associated Cache and Set Associative Cache.

The main differences between each *Cache Placement Policy* are the way each memory address is stored into their respective cache blocks and how the metadata is categorized.  However, what each *cache placement policy* has in common is how they are sized.  Normally, each cache will be assigned a certain block size and consist of a certain number of blocks.  Each block will also be assigned a particular size as well.  The cache block will be able to hold multiple memory addresses depending on the block size.  For example, if a block consists of 4 bytes, and each memory address contain 8 bits, then that means that the block will be able to store 4 memory address.  If that particular cache holds 16 blocks, then that cache will be able to hold a total of 64 memory address (16 \* 4).  This particular example pertains to Fully Associative and Direct Mapping, but differs from the Set Associative Cache.  In addition, each cache will contain different pieces meta data used to identify each individual memory address.  The three main types of data used are tag, index and offset.  The tag will contain the beginning numbers of the memory address, the index will contain any numbers in the middle, and the offset will contain the last numbers of the memory address.  This criteria is used to locate the memory addresses stored within the cache.

A **Direct-Mapped Cache** is a simple cache architecture where each block of main memory maps to exactly one cache line based on the index bits in the memory address. The memory address is divided into a **tag**, an **index**, and an **offset**. The index determines which cache line to check; the tag is used to verify if the desired block is present; and the offset specifies the exact byte within the block. Only **one memory block** can reside in a cache line at any time, so if two addresses map to the same index, the new block **replaces** the old one, regardless of how recently it was accessed. This simplicity makes Direct-Mapped Caches easy to implement and fast, but they suffer from high conflict misses, which can reduce the hit rate compared to more flexible cache designs.

A Fully Associative Cache is similar to a Direct Mapped Cache in that it has the capability of storing an address or multiple address in a set of cache blocks.  However, the criteria used to store the memory addresses are different.  For a Fully Associated Cache, each memory address will be stored into a cache block based on its tag number.  Multiple memory addresses can be stored into a cache block as long as they share the same tag number.  In addition, the metadata utilized in a Fully Associated Cache is slightly different than the metadata used in a Direct Mapped Cache.  In a Fully Associated Cache, the index number is nonexistent, and there is another piece of metadata used known as the LRU.  The LRU (least recently used) is used to keep track of which cache block was most recently used.  Therefore, each time a cache block is accessed, the LRU is automatically updated.  The remaining metadata tag (size of memory address bits – offset) and offset (log2(size of block)) remain the same as before.  The advantages to utilizing Fully Associative Cache is that it has a higher hit rate because there’s more flexibility involved, and each cache block isn’t depending on the index number.  In other words, the only time a memory address is replaced is when the new memory address shares the same tag number as the least recent memory address.  The disadvantages associated with a Fully Associated Cache is that it has slower performance because it takes long to traverse through each cache block, similar to how a linked list is operated.  In addition, this method is the most expensive compared to the other cache placement policies*.*

A Set Associative Cache could be considered as a cross between a Direct Mapped Cache and a Fully Associative Cache.  Similar to a Direct Mapped Cache, multiple memory addresses are inserted into a specific cache block based on their index number.  However, unlike a Direct Map Cache, the Set Associative Cache will keep track of the tag number of each memory address contained in the cache block.  Therefore, if the cache is attempting to insert a new memory address into an occupied cache block, the least recent memory address will be replaced.  In a cache with 1B cache block, the metadata is structured similar to a Fully Associative Cache with the index (log2(size of cache – size of block) - 1), Way A Data (contains a the first memory address), Way A Valid, Way A tag (number of bits – index - offset), LRU (keeps track of the most recent memory address inserted into the cache), Way B Data (contains the second memory address), Way B Valid and Way B Tag.  The advantages of a Set Associative Cache is that it contains the advantages of both the Direct Mapped Cache and the Fully Associative Cache.  In addition, the Set Associative Cache has the ability to replace one memory address at a time without replacing the whole entire cache block.  However, the disadvantage of the Set Associative Cache is that every cache block will not always be fully occupied.

Overall, cache is very beneficial to use because it prevents the CPU from constantly having to travel back and forth to the main memory to collect data.  As a result, this significantly enhances performance time because the CPU is more likely going to have access to the information it needs.  The main advantage to having cache is that it allows the CPU to access data faster than being forced to retrieve the data from main memory.  As a result, this increases the computer’s performance time.  However, the disadvantage to having cache is that it can be very pricy, and normally you won’t be able to store much information on it because it is quite compact.

From my research, I can conclude that the cache is a very beneficial tool that will significantly enhance the performance of the computer.  The metaphor I would use to describe cache is preparing for an apocalypse.  Let’s say one day you decide to purchase extra amenities such as food and toiletry items, then store them in your attic in case for future use.  As a result, this will make your life more convenient because if an apocalypse were to occur, you wouldn’t have to drive back and forth to the store to purchase the necessary items because you would already have them present.  In my opinion, this is the same principle for a Cache, because it allows the CPU to have access to what it needs rather than being forced to inconveniently retrieve the data.

Bibliography:

1. Tanenbaum, Andrew S. and Austin, Todd, “Structured Computer Organization”, 1975
2. Gapo, Branko, “What is CPU Cache? (L1,L2, and L3 Cache)”, September 29, 2022

https://cpuninja.com/cpu cache/#:~:text=These%20terms%20denote%20the%20multileveladd,it%20is%20costly%20to%20make

1. Areej, “Difference Between L1,L2, and L3 Cache: How Does CPU Cache Work ?”, December 4, 2021

<https://www.hardwaretimes.com/difference-between-l1-l2-and-l3-cache-how-does-cpu-cache-work/>

1. Phillips, Gavin, February 17, 2021, “How Does CPU Cache Work ?  What are L1, L2, and L3 Cache ?”

https://www.makeuseof.com/tag/what-is-cpu-cache/#:~:text=The%20L1%20cache%20is%20usually,operation%20is%20to%20be%20performed.

5) Lutkevich, Ben, Unknown Date, “Cache”

<https://www.techtarget.com/searchstorage/definition/cache>

6) Unknown Author, Unknown Date, “Cache Placement Policies”

<https://encyclopedia.pub/entry/33147>

7) Unknown Author, Unknown Date, “Cache Memory”

<https://www.toppr.com/guides/computer-science/computer-fundamentals/primary->memory/cache-memory/