RUNNING HEAD: Cache Memory

Cache Memory

Aidan Hopper

ID: 42236214

CS 312

Dr. Donald Davendra

I pledge that this submission is solely my work, and that I have neither given,

nor received help from anyone.

**Abstract**

Caches are a specific type of memory in a computer which acts as a transitionary layer from secondary storage to the processor. In modern day computers there are multiple caches designed in a hierarchy with the top of the hierarchy being the processor. The cache hierarchy starts at the top, which is the L1 cache, and directly communicates with the processor. The next cache is the L2 cache which communicates with L1. So the hierarchy goes L1, L2, L3… and so to the bottom of the hierarchy, which often times is the disk. There are three techniques that are used to perform the mapping of data from a higher level cache to a lower level cache, this is refereed to as the mapping function. The three techniques are direct mapping, associative mapping, and set associative mapping.

Table of Contents

**Introduction**

The concept of cache is what allows for high CPU efficiency, first introduced in 1985 with IBM’s Model 85 super computer as a way to massively increase CPU efficiency due to an increase memory transfer speed to the processor (Pugh et al., 2003). Cache enables computers to process more requests because of the increased transfer speeds up the memory hierarchy. This paper will explore cache memory in depth, looking at the concept of memory, cache, the cache hierarchy, and how caches work on a technical level.

**What is Memory?**

One of the prerequisites to understanding a computer cache is to understand the basic concept of memory. One of a computers most basic functions is to be able to store data, be that on a disk or in other areas of the system. The issue with accessing this data is that the transfer rate is slow, causing latency in the computer. The solution to this problem is to implement a faster temporary memory to store data that is in immediate use. This temporary memory is referred to as cache.

There are two types of memory, the first is primary storage. Primary storage makes up the computers caches, and is volatile, meaning the memory is lost when the computer loses power. The data apart from the instructions stored in the processor is temporary. The next type of memory is secondary storage which is non-volatile and designed for permanent storage. Examples of secondary memory is a disk (*Secondary Storage - Storage - OCR - GCSE Computer Science Revision - OCR*, n.d.).

**What is Cache?**

Cache memory enables computers to process data from slow transfer rate storage quickly and efficiently. This is because with memory there is a trade off between size and speed. This means that the fastest transfer rate memory is also going to have the smallest amount of space. Essentially the cache enables the system to process a larger amount of requests than without a cache. Before the use of cache memory processors would idle and efficiency was dramatically reduced which is why all processors today have cache memory.

**Cache Hierarchy**

Because of the trade-off inherent to data storage being size for speed, it means the optimal way to send data from a large capacity drive to processing is by using a hierarchy of cache with each level being faster and smaller than the other. These levels are referred to L1, L2, L3…, all the way to main memory, which has the largest capacity. The L1 cache is directly accessed by the processor, and is the fastest memory in the whole system, while also being the smallest. Another trade-off that hasn’t been previously mentioned is the cost, faster memories cost more per bit. The L1 cache then accesses the L2 cache for data and so on, all the way to RAM, which directly accesses the disk.

On a typical system, each core in the CPU will have two L1 caches, for data and instructions respectively. L1 caches are very small with their size typically being around 100KB, but can widely vary depending on the processor.

Figure 1 showcases the hierarchical nature of multi-level caches, with the L1 cache communicating with the processor’s registers, L2 cache communicating with the L1 cache, and so on. One important note is that registers are not considered a cache, this is because the registers are at the top of the hierarchy, meaning it is the final destination for all data to be processed. Another way to explain it is that there is nothing pulling data from the register. Another important note is that the bottom of the hierarchy, which in this case is the disk, is also not a cache because it is the source of the data

**How Does Cache Work?**

When cache receives a read request from the processor it is requesting a block, which is a range of memory addresses, of memory to be read. The block that is being requested is transferred into the cache. Then when a program tries to reference a location in the cached block, the content is read from the cache instead of main memory. There are three main techniques for deciding where the blocks are stored on the cache which are called mapping functions. The first and simplest one is direct mapping (Parthasarathi, 2018).

**Direct Mapping**

For the direct mapping technique, block I of main memory is mapped to j mod (number of blocks in cache. For example if the cache has a 16 block capacity then the first 16 blocks in main memory are mapped to indexes, 0, 1, 2.. to 15. The obvious issue with this approach is that if the processor makes a request to block 0, then to block 16, then a collision will occur, causing block 0 to overwritten, even if there is still space in the cache at any other index. A way to address this issue would be to make the cache larger, reducing the probability of collisions, but this does not address the underlying design issue (Parthasarathi, 2018).

**Associative Mapping**

Another technique for performing the mapping operation is associative mapping which is more complicated than direct mapping. With associative mapping a block can be placed into any block position on the cache. To achieve this when a processor is searching for a block in the cache, it has to search through the entire cache to see if the block is present, and will only replace a block if the cache is full. The issue with associative mapping is that there is a performance cost associated with it, compared to direct mapping with only requires one lookup to find a block (Parthasarathi, 2018).

**Set Associative Mapping**

The last technique that will be discussed in this paper is set associative mapping, which is considered a compromise between, associative mapping and direct mapping. With set associative mapping, blocks in the cache are grouped into into sets *k* number of blocks. This allows a block retrieved from main memory to be stored in any block of its specified set. This method allows for a lookup to be done in *k* time, instead of having to search through the entire cache. The flexibility of associative mapping is also utilized through sets While set associative mapping doesn’t completely remove the possibility of a collision occurring, it drastically reduces it.

**Conclusion**

Caches are the concept of hierarchical memory that allow computers to process data efficiently. The cache hierarchy allows data from secondary non-volatile storage to be processed by the CPU. The most essential aspect of caching is the process of retrieving data from lower down on the hierarchy, the function that performs this action is called the mapping function. There are three types of mapping functions, direct mapping, associative mapping and set associative mapping, each with their own costs and benefits. Caching has allowed computer to alleviate memory transfer speed bottlenecks to get the most performance out of the processor.