Cache Memory Introduction and Analysis of Performance Amongst SRAM and STT-RAM from The Past Decade

Carlos Blandon
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816-2362

Abstract—The following paper will be an analysis of performances between a variety of cache designs, new and old. We will discuss the basics of cache memory, cache design, and the handling of data amongst the CPU, memory and cache. We will then dwell into past designs and compare and contrast them to newer technology. We will test cache latency as well measure their energy consumption. A special interest will be the multi-level cache designs with a level 3. We will explore these multi-level cache designs and expand on their purpose and their effectiveness compared to non L3 models. The main focus of the paper will be trying differentiate the pros and cons between SRAM and STT-RAM when it comes to energy consumption and cache latency.

Keywords—STT-RAM, SRAM, L3 Cache, CPU, Cache latency, Energy Consumption, Volatile memory, Non-Volatile memory, CMP, Cache modeling, Multithread, Cache coherence.

I. INTRODUCTION

Cache configurations vary from system to system depending on what is required, or the functions that the selected system is performing. A small system might require speed while a system might require more memory and disregard the speed and finally it might require a balance of both. Cache memory is used to store data that could easily and quickly be retrieved instead of storing it into the main memory and retrieving it from the main memory.

While there are many different configurations in cache the ultimate goal of cache is to help the flow of information by making the task much faster than it would be by using the main memory. Cache memory’s advantage is its placement which is quite close to the CPU, which requires less pathing in the hardware. We will explore the placement of cache and the overall strategy of its placement much further.

As stated before there are multiple cache configurations, these configurations depend on the system build and overall what the user wants to accomplish. Before we talk about configurations we need to discuss the different levels of cache itself. The simplest type of cache would L1 cache or Level 1 cache. This cache is the closest to the CPU, it is quite fast but also has a small memory. Level 2 or L2 cache follows L1 cache, this cache is a bit smaller but has a larger memory space. Finally, we have L3 cache (Level 3 Cache) which is a link between both L1 and L2 caches and helps the transfer of data amongst both caches. [2]

Having these different types of caches is important because there are different needs of memory per task. Also by having these multiple levels of cache means you configure systems in a variety of ways to suit your needs. If you are in need of a fast response, then you would definitely want a very efficient L1 cache and maybe an L2 cache just in case you need a bit of space. If you wanted a more balance cache you would definitely want L1, L2, and L3 just to be sure that the memory flow amongst all cache will be optimal. Now this might be an oversimplification of the methods but it should point out that energy efficiency is also a factor in this. You would not want a system that takes too much energy making a system not optimal.

Cache configurations can vary but there are some commonly known block placements such as direct mapped cache blocked placements, full-associative cache block placement and set-associative cache block placements. These cache block placements have their unique structure and uses that are applied in most systems.

Cache memory is not only readable but it can sometimes be storable past its use and even after the system is powered down. This sort of memory is called non-volatile memory which means that even after there is no power the data will be there once there is power again.[7] Examples of this sort of memory is a characteristic of STT-RAM. On the flip side there is memory which will not work once the power is in the system meaning that the data will be erased once powered down. This is how SRAM and eDRAM memory work. There are cons and pros for having both of these types of memories, and again it is up to the developer to pick which will best suit their needs.

How these different memories are accessed all depend on how they are mapped. Direct mapped cache used the lower bits of the tag to access the directory or memory.[5] The upper bits ensure whether there is a hit or a miss when retrieving data. In a set-associative cache you can think of it as a direct mapped cache multiple times. In a direct mapped cache, you have one tag and one data segments, set associative has multiple of both, tag and data. You can acquire more parallelism though set-associative cache than direct mapped cache.
Going deeper into the hardware of a cache we could open up a theoretical cache and see how it is built and functions from a hardware perspective. Cache memory is broken down to multiple lines, the number of lines depends on the cache memory. These line contain a tag and a data segment. Data segments store the value that was retrieved from the main memory. The tag segment of the lines contains the directory or the location of where on the main memory is the data retrieved from. [9]

With relation to cache, data retrieval is not always 100%, this is where the terms miss ratio and hit ratio are established. A miss ratio is the ratio of how many times data was trying to being accessed by the cache but there was no data at the destination. When the data is not where it is supposed to be in the cache then it is retrieved in the main memory, placed on the cache memory then it is accessed. Having a low miss ratio means that you have faster response times since the cache isn’t retrieving much memory from the main memory.[4] The hit ratio describes when cache data is being retrieved at cache and the data is there, unlike when there is a miss. When there is a hit in cache then there is no search for the data in main memory. Having a high hit ratio is ideal because it means that data retrieval is optimal requiring less steps to attain the data.

Further into my analysis we will see the growth in speed and efficiency amongst cache memory in the past decade and a half. I will compare different specs from DRAM, SRAM and STT-RAM to get a broader view of the improvements over time. We will touch base on the newer technology that has impacted cache memory and its drastic efficiency upgrade over the past decade.

II. LITERATURE REVIEW

There are many sorts of cache designs that all have their use. These uses could vary from the amount of memory available, how fast the memory is accessed or how long we could retain data before it is lost. There is an L1 cache design called the nonvolatility -relaxed STT-RAM.[3] This cache design sole purpose is to be able to hold data for a certain amount of time. The data retention will help in the overall performance of the data retrieval process. You would want to retain data because you want cache to be filled with frequently accessed data so if you were to hold data a bit longer than normal then that data will not leave cache that extra microsecond and will used again. This is an improvement of microseconds but over time the amount saved could be exponential.

STT-RAM is very advantageous compares to static RAM. Some of it advantages its energy consumption, it is quite low compared to regular static RAM. Although this technology is quite popular it is riddled with a lot of errors that makes it a questionable choice at times. The size of the RAM is usually small and the smaller we get with hardware the harder it is for use to handle it and its errors in the fabrication process. [10]

Only recent have we been able to use last level cache. Since most systems before could barely handle two caches or in better sense didn’t have the need to use it. Once our systems started to get more complicated L3 cache is becoming something desired for our needs. In order to get the most of our caches there is a method of prefetching data to our cache memory. This sort of mechanism are quite useful because memory is there before it is even needed, which for memory retrieval purpose is handy. The only downsides of this is that you have to increase the amount of data write functions which could potential cost too much energy to be efficient when energy consumption is a priority. [9]

There are ways to also achieve ultimate energy reduction from cache. Of course such a method is using STT-RAM dude to its low energy consumption. The CCear method reduces the amounts of time the cache has to refresh to get new data, by doing this you save that extra instruction. Of course this sort of cache configuration is much more complicated than the simple explanation given. There are many intricate parts that play a big role to achieve low energy consumption, CCear is just proof of how the advantages of STT-RAM are being used to improve overall performance in data retrieval when it comes to cache hits.[8]

III. DATA ANALYSIS

Fig. 1. Cache modeling showing how the cache latency for different models varies and how it has gotten significanlently smaller over time.
IV. CONCLUSION

From these various I noticed that the advantages of STT-RAM over normal statin RAM are quite large. Even though STT-RAM is advantageous I feel as though sometimes its benefits are on a knife's edge, it could either help the system dramatically or wear it down to the point were static RAM memory would be more appropriate. These studies also show how the field experts keep on pushing for more STT-RAM configurations and keep advancing L3 cache.

REFERENCES


## TABLE I. SYSTEMS CONFIGURATION

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Khoshavi [2]</td>
<td>8</td>
<td>3GHz</td>
<td>32KB</td>
<td>8-way</td>
<td>SRAM</td>
<td>512</td>
<td>MESI</td>
<td>512KB</td>
<td>8-way</td>
<td>SRAM</td>
<td>8192</td>
<td>MESI</td>
<td>96MB</td>
<td>16-way</td>
<td>eDRAM</td>
<td>~100M</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>Sun [3]</td>
<td>4</td>
<td>2GHz</td>
<td>32KB</td>
<td>4-way</td>
<td>SRAM</td>
<td>512</td>
<td>N/A</td>
<td>256KB</td>
<td>8-way</td>
<td>SRAM</td>
<td>4096</td>
<td>N/A</td>
<td>4MB</td>
<td>16-way</td>
<td>STT-RAM</td>
<td>65536</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Li [9]</td>
<td>16</td>
<td>2GHz</td>
<td>32KB</td>
<td>2-way</td>
<td>STT-RAM</td>
<td>512</td>
<td>WB</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>8MB</td>
<td>16-way</td>
<td>STT-RAM</td>
<td>131072</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>Zhang[10]</td>
<td>16</td>
<td>3.5GHz</td>
<td>32KB</td>
<td>4-way</td>
<td>SRAM</td>
<td>512</td>
<td>MESI</td>
<td>256KB</td>
<td>8-way</td>
<td>SRAM</td>
<td>4096</td>
<td>N/A</td>
<td>16MB</td>
<td>16-way</td>
<td>SRAM</td>
<td>262144</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Mao[8]</td>
<td>4</td>
<td>4GHz</td>
<td>32KB</td>
<td>4-way</td>
<td>N/A</td>
<td>512</td>
<td>N/A</td>
<td>256KB</td>
<td>8-way</td>
<td>N/A</td>
<td>4096</td>
<td>WB</td>
<td>8MB</td>
<td>16-way</td>
<td>STT-RAM</td>
<td>131072</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>Joo[7]</td>
<td>1</td>
<td>2GHz</td>
<td>32KB</td>
<td>N/A</td>
<td>SRAM</td>
<td>512</td>
<td>WB</td>
<td>8MB</td>
<td>16-way</td>
<td>PRAM</td>
<td>131072</td>
<td>WB</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Yazdanshenas [6]</td>
<td>4</td>
<td>1GHz</td>
<td>32KB</td>
<td>4-way</td>
<td>DRAM</td>
<td>512</td>
<td>MESI</td>
<td>2MB</td>
<td>16-way</td>
<td>STT-RAM</td>
<td>32768</td>
<td>WB</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Jokar [5]</td>
<td>4</td>
<td>3GHz</td>
<td>32KB</td>
<td>8-way</td>
<td>DRAM</td>
<td>512</td>
<td>MOESI</td>
<td>2MB</td>
<td>8-way</td>
<td>STT-RAM</td>
<td>32768</td>
<td>WB</td>
<td>8MB</td>
<td>8-way</td>
<td>ReRAM</td>
<td>131072</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>Chang [4]</td>
<td>8</td>
<td>2GHz</td>
<td>32KB</td>
<td>8-way</td>
<td>N/A</td>
<td>512</td>
<td>MESI</td>
<td>256KB</td>
<td>8-way</td>
<td>N/A</td>
<td>4096</td>
<td>MESI</td>
<td>32MB</td>
<td>16-way</td>
<td>N/A</td>
<td>524288</td>
<td>WB</td>
</tr>
<tr>
<td></td>
<td>Crawford[1]</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

"CL" = Cache line

**Calculation for "# of CL" columns:**
Manually compute the number of cache lines given the capacity value as listed in capacity column, assuming the cache line size is always 64 Bytes

**Protocol column = {Write Back (WB), Write Through (WT), MESI, MOESI, Not Available (N/A)}**