Comparison of Energy and Performance Efficiency of Recent Cache Configuration Trends and Designs

Mohammad Hammad
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816-2362

Abstract— Over the past ten years cache configurations and their designs have increased in importance, energy efficiency, speed, and in other aspects as well. The goal of this paper is to discuss the energy and performance efficiency of different cache designs throughout the years. A good scale efficiency for a cache could be considered a measure of its latency and energy consumption. To optimize cache efficiency, a CPU designer should focus on the RAM type and the strategy being used to determine cache block placements. The paper will illustrate the similarities between STT-RAM, SRAM, eDRAM when considering their individual effects on energy consumption and latency. As the demand for technology grows so does the demand for energy consumption, therefore over the past ten years the use of STT-RAMs increased as they use up less energy. More designers are also pushing towards SRAM over eDRAM as even though it is more expensive, it is much faster.

Keywords— Cache Levels, Non-Volatile Memory, Volatile Memory, Cache Latency, Memory Hierarchy, SRAM, STT-RAM, eDRAM, Cache Lines, Direct Mapped, full-associative, set-associative, Miss Ratio, Hit Ratio, Energy Consumption, Cache Block, associativity

I. INTRODUCTION

Cache configuration is one of the most effective ways to design an energy and speed efficient processor, as it is one of the most crucial levels in memory hierarchy. The optimal cache configuration will exhibit locality, multiple levels, and relatively effective cache block placements to ensure fast memory access. As memory hierarchy explains, items with less memory process faster therefore by using cache to access memory locations rather than main memory the processor can execute faster. However as cache has less memory, we must use spatial and temporal locality to determine which data from our program should be stored as blocks in our cache memory or cache lines. Considering the numerous functions a cache must perform like access main memory, update cache lines, store/output data, etc.; the use of multilevel caches is increasing in popularity due to the increase in efficiency it delivers.

Multilevel caches are increasing in popularity for numerous reasons. However mainly due to its ability to increase cache memory and coordinate multi-core processors. Every core has its own L1 cache which allows it to independently read/write. However as multi-cores must work simultaneously, they must also share some storage. The L2 cache and L3 cache are used as shared storage for each core, with L2 being faster but L3 having more memory. Memory in processors are always randomly accessible and have read/write capacity. They could also be volatile or non-volatile, volatile memory requires voltage supply to maintain values. Most memories are volatile as they use either transistors or capacitors for storage, for example SRAM and DRAM respectively. STT-RAM is a non-volatile memory as it uses spin transfer torque and magnetism to determine active elements.

In order to reduce a cache’s miss ratio, we can increase its associativity by approaching a design which will provide flexibility in deciding which block of memory can be associated with a corresponding line in cache. There are three common design strategies used to determine cache block placements: direct mapped, full-associative, and set associative. Direct mapped doesn’t use associativity, so each block is mapped to only one possible line. This allows us to used fast bitwise mod of addresses but will likely decrease our hit rate. Fully associative designs have unrestricted associativity, meaning blocks could be stored in any cache lines. This design gives our cache full flexibility in all of its memory, however it has a large tag field making it take longer to search for a block. Set associative designs exhibit bounded associativity and allow a dependent value of lines to store each block. For example a 3-way set associative gives every block of memory 3 possible cache lines. This helps us balance flexibility with tag matching complexity but some conflict misses still occur.

Unrestricted associativity designs use direct-map to access the contents of a cache. To access contents of cache you can use bit field encoding. The total encoding will represent the tag, cache line, word size in bytes, and block size in words. These 4 encoding combined will form the address of the content in cache. The tag for a cache content in direct mapping is equal to the number of cache lines, byte size of words, and amount of words in a block all subtracted from the main address bit. Set associative cache designs operate similarly except instead of storing the number of cache lines, we represent number of sets in the cache. The number of sets in the cache is determined by dividing
our total cache lines by k, which is dependent on the k-way set associative desired. Due to the way cache performs and bit encodes in associativity designs, its structure is similar to memory. Cache designs also need one-way buses from the CPU and to the main memory, control inputs from the CPU and to the memory, bi-way data buses from the CPU and to the main memory, and hardware for a 2-bit counter is included with each cache line.

---

**Fig. 1.** Three level cache configuration.

Hit ratio and miss ratio are greatly significant to the efficiency of a cache configuration. Hit ratio is the portion of memory accesses which can be found in the cache and the miss ratio is the portion of memory accesses not found in the cache. When the CPU needs to perform any operation it needs to access the byte address of memory for all parts of that operation, memory hierarchy uses locality to store these addresses in cache. Therefore as the CPU requests memory accesses, a cache that has them already stored can quickly executed the operation and return the value to the CPU. So if a cache design has a high hit ratio it is likely to perform efficiently.

Over the years cache designs have been improving in efficiency, due to research and experimentation with various design aspects like device technology, set association, etc. In Section 2, we compare the efficiency of different cache designs over the past ten years in order to conclude the advantages of various design features.

### II. LITERATURE REVIEW

#### Cache Configurations from 2006-2011:

As technology progresses and the amount of multicore architectures in processors increase, so does the demand for on chip cache memories. Due to this trend STT-RAM started getting more popular after 2006. Spin-Transfer Torque (STT) is a promising candidate for future multi-core general purpose and embedded system. Due to it’s high-density, low leakage, and immunity to soft errors STT-RAM uses less energy and performs faster than a SRAM [1]. Although STT-RAM has a high write latency and dynamic write energy, SRAM has gotten increasingly popular due to STT-RAM. The results show that although we have an increase in dynamic energy, we observe improvement in total energy mainly due to the drastic reduction in leakage energy.

Due to the importance of multiple cache levels, throughout the years different RAM types have been used depending on cache levels. Due to the rising need for energy efficiency and desire of low latency, designers use different RAM types on different levels. Each level is faster than the next but holds less memory. Therefore the lowest level of caches usually hold SRAM, as they must perform the quicker than the next level in memory hierarchy in order to keep the cores coordinated. Cache capacity came as a good, in 2006 a study used a set of 32 KB, 256 KB and 64 MB for L1, L2 and L3 level caches and used DRAM for both cache level 1 and 2 [2]. This was quite common as most cache configuration stopped at level two. Five years later a study used the same test plan for STT-RAM cache designs; they too had 3 cache levels with 32 KB, 256KB and 4MB capacities but final level cache was STT-RAM while the others were SRAM. Again, the purpose of using different RAMs on different levels is to optimize all aspects of our performance. For instance while the SRAM in L1 and L2 operated faster, it used more energy when compared to the STT-RAM in L3. Although using a STT-RAM increased latency, the impact the energy conservation had is considered more efficient. SRAM is just as fast as eDRAM but uses way less energy, thus in some configurations SRAM completely replace DRAMs. For example in 2013 a processor with a 3 level cache system, 32KB SRAM for the first level, a 256KB STT-RAM for the second level and a 32MB eDRAM for the third level. This resulted in a latency period of 3.05 ns for STT-RAM and 4.46 ns for SRAM, as well as energy consumption of .94 nJ for STT-RAM and 2.10 nJ for SRAM [4]. In this case STT-RAM is non-volatile so it consumes less energy, exhibiting that the configurations of cache depend specially on the functions they’d perform.

#### 2011-2016 Cache Configurations:

As we approach the present, cache configuration lean more towards STT-RAM, SRAM, and eDRAM with higher cache capacities at higher cache levels. Due to STT-RAM low energy consumption and fair speed, more designs are configuring a STT-RAM at L2 of the cache. One study which focused on the vulnerability of eDRAM led to use of a 32KB L1 on a SRAM, 1MB STT-RAM on L2, and an 96MB eDRAM for L3[5]. This follows the trend of memory hierarchy since as you go up in cache levels your decrease speed and increase in capacity. A different study emphasizes the overwrite energy efficient of STT-RAM, which helps explain why they are increasing in demand. On a 2 level cache with a 32KB L1 on a SRAM and a 1MB L2 on a STT-RAM, the paper explains the values for latency and energy consumption whilst using the WB protocol. The cache proved to produce a latency of 9.08 cycles on STT-RAM, 6.03 cycles on SRAM and consumed .216 nJ of energy on STT-RAM and .156 nJ energy on SRAM [6]. This concept of keeping the more speed and energy efficient caches on lower levels is evident throughout cache configuration designs.
III. DATA ANALYSIS

As the functions and capabilities of technology evolve, their components must evolve as well. Most computer systems and trending towards using multi-core processors which ultimately requires multi-level cache configurations. As the multiple cores work simultaneously, they must exchange data at one point or another. In cache configurations most of this communal data sharing occurs in L2 of the cache. L1 is always used to perform read and write operations and finally L3 is used for extra storage and coordination between processors. As time has progressed cache configurations have advanced to execute faster and conserve more energy, while essentially using the same amount of space. It is evident that the most important factors in scaling cache configurations are speed and energy consumption.

IV. CONCLUSION

REFERENCES


| Khoshavi [5], 2016 | 8 | 3GHz | 32KB | 8-way | SRAM | 512 | MESI | 512KB | 8-way | SRAM | 8192 | MESI | 96MB | 16-way | eDRAM | ~100M | WB |
| Chen [6], 2016 | 4 | 3.3GHz | 32KB | 8-way | SRAM | 512 | WB | 4MB | 8-way | STT-RAM | 65536 | WB | NA | NA | NA | NA | NA |
| Khoshavi [7], 2016 | 8 | 3GHz | 32KB | 8-way | SRAM | 512 | WB | 512KB | 8-way | SRAM | 8192 | WB | 96MB | 16-way | eDRAM | ~100M | WB |
| Lin [4], 2015 | 2 | 800MHz | 32KB | N/A | N/A | 512 | NA | 512KB | NA | NA | 8192 | NA | NA | NA | NA | NA | NA |
| Jog [1], 2012 | 4 | 2GHz | 32KB | 4-way | SRAM | 512 | WB | 4MB | 16-way | STT-RAM | 65536 | N/A | N/A | N/A | STT-RAM | N/A | NA |
| Jaleel [2], 2006 | 8 | NA | 32KB | 4-way | STT-RAM | 512 | WT | 256KB | 8-way | DRAM | 4096 | WB | 4-64MB | 16-way | DRAM | 65536-100M | WB |
| Sun [3], 2011 | 4 | 2GHz | 32KB | 4-way | STT-RAM | 512 | MESI | 256KB | 8-way | SRAM | 4096 | WB | 4MB | 16-way | SRAM | 65536 | WB |
| Chang [4], 2013 | 8 | 2GHz | 32KB | 8-way | SRAM* | 512 | MESI | 256KB | 8-way | SRAM* | 4096 | MESI | 32MB | 16-way | SRAM* | 524288 | MESI |

"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)}