STT-RAM vs. SRAM/eDRAM and Efficiency Analysis between Differing Cache Configurations

Brandon Dziewior
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816-2362

Abstract—This paper looks at the different comparisons between gathered cache configurations across multiple research topics. Understanding that there are key metrics such as the set association, device tech, and capacity help determine the optimal configurations for when dealing with a multitude of different cache types. The amount of cores and the frequency that the CPU runs at are also relatively important when dealing with which cache configuration would be best for each particular system design. One such instance of an interesting cache design would be that from Zhenyu Sun’s paper, illustrating a hybrid STT-RAM design for use in saving energy consumption. Another such design that seemed particularly interesting was Mu-Tien Chang’s design on a third-level eDRAM cache to show the benefits in comparison to an STT-RAM or SRAM configurations on the third level.

Keywords—STT-RAM, SRAM, eDRAM, Volatile/Non-Volatile Memory, Multilevel Caches, Cache Coherence, Multiport Systems, LLC, MLC, FPGA, HDL, ASTRO, NUCA

I. INTRODUCTION

When looking through the different baseline and additional reference papers they gave an insight on the importance of cache configuration. Since the use of multiprocessors has become more prevalent as the years have passed on, a difficulty faced with is the identification of data between the different processors and the main memory within the system. To sort out the underlying issue of data not being shared correctly across the processors a solution of cache coherence protocols were presented to help alleviate the disparity [1].

Multilevel caches are a supporting matter and are used in most systems today due to the ability of locating data in a faster manner than before. These multilevel caches are important because without them each processor would rely on a high-capacity and slow cache to process all the information before transmitting it back to the processor. In a multilevel cache system the faster and lower capacity cache level is the first source the processor calls for before proceeding to the slower and higher capacity cache. However, due to first cache designs that are called upon being faster it causes their power to have a higher chance of leaking. Due to this, the slower, but higher capacity cache designs is necessary as backup if the call to the faster cache is a miss [4].

When constructing the cache an important design decision is choosing which type of cache block placement would be best suited for the needed task. Direct-mapping is a placement policy of making the task of the base main memory to address only a single cache block, leading to the cache being much quicker, but it has the possibility to miss more data than if it was a full-associative mapping. However, for full-associative mapping, the time it takes to complete the search for each block is much slower due to the address needing to be mapped to each block. Another instance of a middle ground between the two different policies for mapping would be set-associative, which is when there are a particular amount of blocks per set, leading it to be semi-thorugh and semi-fast [3]. In recent years, new forms of device technology have been introduced that offers tradeoffs on its stability among others. Spin-Transfer Torque Random Access Memory (STT-RAM) is one such device that’s primarily non-volatile with its counterparts Static Random Access Memory (SRAM) and Embedded Dynamic Random Access Memory (eDRAM) being known as volatile alternatives [4].

In direct-mapped cache strategies the address checks the index within the cache, and if that index contains data under the correct tag then it is a hit and continues on. If the address comes across an index without any data then it is a miss and the data is not received. For set-associative cache strategies the same fundamental process is completed but it goes through multiple checks within each block dependent on how many ways the set-associative mapping is mapped. So for a given block the address can undergo an n-amount of times to check to make sure the data is a hit. Caches contain an I-Cache and D-Cache, I-Cache to be used for the transfer of instructions, and D-Cache to be used for the transfer of data. When dealing with the different memory components and what is used to store data and receive it, SRAM uses transistors, and DRAM uses capacitors [11].

Processor Chip

Main Memory

Tag

Data

CPU

Copyright © 2023 VLDB Endowment. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than VLDB Endowment must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@vldb.org, located at VLDB Endowment, 6109 Anderson Mill Road, Suite 400, Austin, Texas 78729, USA.
Hit ratio and miss ratio are terms used to better explain the performance of individual caches and their mapping techniques. When constructing a cache and choosing the type of mapping that the hardware will follow it’s important to know these ratios and how mapping will affect the calls that are distributed back and forth between the CPU and main memory. Hit ratio is another name for hit rate, which is how many times the memory access was found within the cache divided by the amount of reference times looking for these memory accesses. The higher the hit ratio the better the design for that particular cache. On the other hand, also take into account the miss ratio which looks at the amount of times the memory access was not found within the cache. To calculate miss ratio you would look at the amount of hits obtained for that design subtracted by one, then divide it all by the amount of times the memory tried to call for the data [11].

Within this paper different cache configurations will be researched and identified, and the benefits of particular design choices over others. Looking at configurations past the year 2000 is the main focus of this research to better grasp an understanding of how design systems have shaped in the recent years in relation to better understand how to optimize memory and the processors that use them. Data gathered from each paper will also be used to graph based on configurations designed by the coordinating writers and to see how efficient they are in correspondence with each paper reviewed.

II. LITERATURE REVIEW

Starting my findings based in 1994, the analysis of different cache protocols was a main focus done within Scott E. Crawford and Ronald F. DeMara’s paper on multiport systems. The paper consisted of comparison between different protocols to see which one demonstrated optimally and was scalable for larger systems. When looking through the different protocols they came to the conclusion of Firefly being the most efficient in terms of time it takes to complete tasks, however, the Directory protocol showed just as much promise due to its scalability when the processor amount was increased [1]. In 2005, Dhruba Chandra and cowriters decided to test the sharing of cache and its productivity between caches. Within the testing phase of their systems and cache contention they chose to use benchmarks that supported more information in relation to the memory aspect of their cache performance. When deciding how to check the sharing between caches and the miss or hit ratios they used a Probability model which proposed to use inductive probability to predict the nature of each cache sharing instance. This model ended up effectively demonstrating a low error percentage versus other models and they obtained the model they designed to be more efficient [8].

In more recent years, starting in 2011, Zhenyu Sun and fellow designers devised up a new design for the increasingly prominent STT-RAM. STT-RAM being looked into more due to its effectiveness in comparison to SRAM and DRAM while maintaining a minimized cache leakage power. The STT-RAM design they chose to come up with was proposed to be used for all levels of cache, instead of using SRAM with or without STT-RAM alongside it. They found their design to help alleviate the energy consumption of previous designs and SRAM caches. They were able to keep the same performance levels all while being able to reduce the energy for their given design in comparison to a previous SRAM/STT-RAM cache design by 73.8% [6]. In 2013, Mu-Tien Chang and designers decided to take a look at the different last level cache alternatives (L3) due to the growing increase for the energy improvement and performance increases that come along with them. They decided to take a look at STT-RAM and eDRAM L3’s in comparison to the traditional SRAM. Effectively, through their conclusions when referencing the different alternatives for last level caches the eDRAM would be the best choice if the refresh was controlled more efficiently [7]. In 2014, Sadegh Yazdanshenas took a look at last level cache performance as well, but his main focus was on the choice of STT-RAM instead of the developing eDRAM. Due to the power consumption of the STT-RAM he surmised that it would be a good solution for last level caches as long as the write issues were maintained to a minimum. The alternatives that he came across when looking at designs for improving STT-RAM cost the performance in energy and a delay in the system response time. The design he and his fellow designers ended up with helped lower the issues revolving around the write latency and write consumption that normal STT-RAM designs came across as well as even improving responses regarding energy throughout the last level cache [10].

The beginning of 2015 marked Mingjie Lin and cowriters findings and design on a processor based solution for FPGA computing. They issued that other alternative computing systems for processors don’t have the complex logic that FPGA units offer as well as the flexibility on top of it. The problem that they recognized with choosing such a system would be the lengthy process of programming for these FPGA units that cost too much expertise that most developers don’t have on the subject regarding HDL programming. Due to this they came up with an ASTRO architecture to help lighten the programming requirements that these FPGA systems require. They were able to maintain the efficiency of the system with their design and still use the FPGA to its full capabilities with an increased use of unused blocks of memory for bridging memory access [5]. Close to the end of 2015 a paper from Mohammad Maghsoudloo became available regarding Non-Uniform Cache Access (NUCA). The look at NUCA was mostly due in part to other forms of RAM such as SRAM, having constant soft errors within the caches. Most designs at the time can’t help alleviate these errors to a great extent and the alternative they choose is NUCA which has a network embedded into the cache. In this paper they were tasked with coming up with a design that used NUCA to certain specifications defined by a space exploration problem. They ended up effectively creating designs with NUCA for their space exploration problem in different objective formats leading L1 and L2 to be able to maintain below the maximum threshold for vulnerability while still using NUCA [9].

In 2016, Navid Khoshavi looked at the vulnerability factor for last level caches that use eDRAM and took apart the vulnerable sequences into different lengths of time for accesses to data that occurred one after another. The main issue he was presenting was in current systems the amount of soft errors in last level caches was increased dramatically due to the density size of the last level cache. He and his fellow designers decided
to come up with a way to test these vulnerabilities on the last level cache (LLC) through a proposed lifetime model. His model was a success and was able to identify where the soft errors occurred the most within the LLC. He proclaimed through the use of his model that following the vulnerability algorithm the most soft errors occur during the medium and long time intervals of the LLC, not so much within the short time interval [2]. Another paper made available in 2016 was in regards to making modern STT-RAM middle level caches (MLC) more efficient in terms of energy. The paper by Xunchao Chen introduced a design that utilizes the high capacity that STT-RAM’s have to offer without the leakage power issues that SRAM maintain. He and his fellow designers came up with the idea of utilizing the emerging technology of STT-RAM due to its powerful capabilities that both DRAM and SRAM have to offer without the potential loss. In the paper they proposed an Adaptive Overwrite Scheme (AOS) to help solve the energy inefficiency within the MLC. They also use Read Distance Prediction (RRD) to help understand the time between a single cache block being accessed multiple times [3]. Lastly, a paper on possible energy enhancement regarding STT-RAM and eDRAM cache levels was created by Navid Khoshavi and designers. A major issue in the current year illustrated by the paper shows how more modern applications that rely on memory usage cause bandwidth issues among different chipsets. The designers proposed using Read Reference Activity Persistent (RRAP) to help cache designs not waste too much energy but still maintain the desired performance. Using both a High Retention and Low Retention STT-RAM with the RRAP in mind they were able to effectively reduce the amount of energy typically required of these STT-RAM designs in memory intensive applications while still maintaining their performance factor [4].

III. DATA ANALYSIS

![Fig. 1. Energy breakdown for STT-RAM, SRAM, and eDRAM between Chen [3], Khoshavi [4], and Chang [7].](image)

![Fig. 2. Latency breakdown for STT-RAM, SRAM, and eDRAM between Read Latency from Khoshavi [4], Chang [7], and Yazdanshenas [10].](image)

IV. CONCLUSION

Looking at the above papers I’ve noticed a few trends to be taken note of such as the increasing usability of STT-RAM and its particular efficiency in comparison to other design systems. I notice that the use of SRAM and DRAM is effective but has its tradeoffs that can negatively affect the system and what you are trying to achieve and how most designers are still trying their best way to effectively use STT-RAM to its full capabilities because of all the benefits it has to offer in comparison. I’ve also noticed how as the years go on the use of more levels of cache have become much more necessary in modern practice due to the sheer amount of data and memory intensive applications people use on a daily basis. When I take a look back at Module 11 I can identify the different designs for memory in general where an SRAM cell requires the use of transistors for it to function while DRAM needs a transistor and a capacitor. I also notice diagrams correctly displaying the way cache is implemented by requiring the address to be sent to the design and based on how the system’s cache is designed determines how long it takes to interpret and store the data.

REFERENCES


---

**TABLE I.**

<WRITE A CAPTION IN YOUR OWN WORDS ABOVE EACH TABLE.>

<table>
<thead>
<tr>
<th>Parameters for the below techniques</th>
<th>Processor</th>
<th>Level 1 (L1) for Instruction (I) or Data (D)</th>
<th>Level 2 (L2)</th>
<th>Level 3 (L3) or Last Level Cache (LLC)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Khoshavi [2]</td>
<td>8</td>
<td>3GHz</td>
<td>32KB</td>
<td>8-way</td>
</tr>
<tr>
<td>Sun [6]</td>
<td>4</td>
<td>2GHz</td>
<td>32KB</td>
<td>4-way</td>
</tr>
<tr>
<td>Chen [3]</td>
<td>4</td>
<td>3.3GHz</td>
<td>32KB</td>
<td>8-way</td>
</tr>
<tr>
<td>Chandra [8]</td>
<td>2</td>
<td>3.2GHz</td>
<td>32KB</td>
<td>4-way</td>
</tr>
<tr>
<td>Khoshavi [4]</td>
<td>8</td>
<td>3GHz</td>
<td>32KB</td>
<td>8-way</td>
</tr>
<tr>
<td>Chang [7]</td>
<td>8</td>
<td>2GHz</td>
<td>32KB</td>
<td>8-way</td>
</tr>
<tr>
<td>Lin [5]</td>
<td>2</td>
<td>800MHz</td>
<td>32KB</td>
<td>1-way</td>
</tr>
<tr>
<td>Yazdanshenas [10]</td>
<td>4</td>
<td>1GHz</td>
<td>32KB</td>
<td>4-way</td>
</tr>
<tr>
<td>Maghsoudloo [9]</td>
<td>16</td>
<td>5GHz</td>
<td>32KB</td>
<td>4-way</td>
</tr>
<tr>
<td>Crawford [1]</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)\}