CONTINUUM

Project Ref. Number ANR-15-CE25-0007

D2.2 – Description of a specific optimization for low power

Version 2.0
(2017)
Final version

Public Distribution

Main contributors:
R. Bouziane, E. Rohou (Inria); and A. Gamatié (LIRMM)

Project Partners: Cortus S.A.S, **Inria**, **LIRMM**

Project Partner Contact Information

Cortus S.A.S Michael Chapman 97 Rue de Freyr Le Génésis 34000 Montpellier France Tel: +33 430 967 000 E-mail: michael.chapman@cortus.com	**Inria** Erven Rohou Inria Rennes - Bretagne Atlantique Campus de Beaulieu 35042 Rennes Cedex France Tel: +33 299 847 493 E-mail: erven.rohou@inria.fr
LIRMM Abdoulaye Gamatié Rue Ada 161 34392 Montpellier France Tel: +33 4 674 19828 E-mail: abdoulaye.gamatie@lirmm.fr	

Table of Contents

1 Summary

While emerging non-volatile memories are a promising low power design solution for modern architectures, they suffer from high write-latency and energy consumption. This makes them less favorable for first level caches such as L1 cache, compared to usual SRAM memory. We propose a compiler-based approach to attenuate the cost of write operations in an architecture that integrates magnetic memory such as the Spin Transfer Torque Random Access Memory (STT-RAM) technology for L1 cache.

In this deliverable, we present an LLVM optimization to reduce the number of silent stores in memory (stores that have no impact because they write a value already present in memory at this location), therefore mitigating the number of write transactions on STT-RAM memory. We show our recent results that confirm the promising impact of this optimization on the energy consumption. We also analyze in details the impact of the application characteristics, the compiler and various microarchitectural features.

Please note that the contents of this deliverable is mainly based on the results published in conferences or journals by the consortium members of the CONTINUUM project. More technical details could be found in the corresponding references.

2 Introduction

Modern high performance Chip Multiprocessor (CMP) systems include large on-chip cache hierarchies, e.g., up to L3 level caches, as illustrated in Fig. 1. An important part of the overall energy consumption comes from the multi-level, on-chip cache hierarchy. In fact, as technology scales down, the leakage power in CMOS technology of the widely used SRAM-based cache gets increased, which degrades the system energy-efficiency. The memory system management techniques offer different ways to reduce their related power consumption. For instance, traditional memory technologies such as SDRAM have the ability to switch to lower power modes upon a given inactivity threshold. Further approaches rather applied to embedded systems, deal with memory organization and optimization [30] [3].

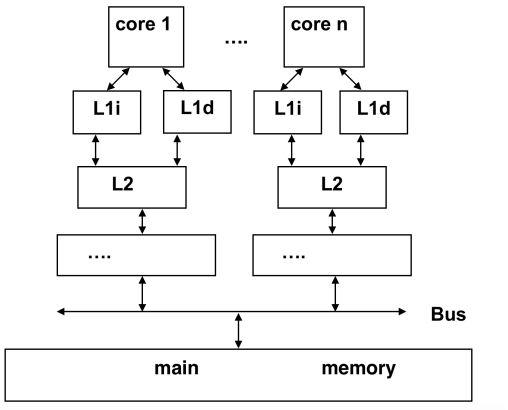

Figure 1: Memory hierarchy model

With the increasing concern about energy consumption in both embedded and high-performance systems, new non volatile memory (NVM) technologies have appeared, such as Phase-Change RAM (PCRAM), Spin-Transfer Torque RAM (STT-RAM) and Resistive RAM (RRAM), that present prominent features and open new opportunities for improving the energy-efficiency of computer systems. Their very low leakage power, makes them very good candidates for optimized energy consumption, as studied in both academia and industry [28]. Yet, NVMs are also suffering from their limited endurance, higher write latency and energy,compared to their counterpart SRAM. This makes NVMs a priori less favorable for write-intensive workloads. Therefore, a straightforward use of NVMs in place of SRAM would inevitably lead to a performance degradation and dynamic power increase that would in turn become energy-detrimental despite the leakage power saving.

NVM technologies have different performance, energy and endurance properties as shown in Fig. 2 from [39]. For instance, PCRAM and RRAM, have limited endurance that is much lower than SRAM and DRAM. However, with current technology, STT-RAM has an endurance similar to SRAM, making it conceivable to design cache hierarchies in non-volatile technologies, including the first level.

There are several works that studied the drawback of NVMs-based architectures using fully-hardware based mechanisms. The most common idea is to manage thoroughly the writes operations with the objective of reducing the number of writes activities on NVMs. Whence, hybrid memory systems where NVM is combined with other types, were rigorously been studied and several approaches were proposed, based on data placement and data migration from NVM to the other type of memory.

Feature	SRAM	DRAM	Disk	Flash	STT-RAM	PCRAM	RRAM
Cell size	>100F^2	6-8F^2	2/3F^2	4-5F^2	37F^2	8-16F^2	>5F^2
Read latency	<10 ns	10-50 ns	8.5ms	25 μs	<10 ns	48 ns	<10 ns
Write latency	<10 ns	10-60 ns	9.5 ms	200 μs	12.5 ns	40-150 ns	10 ns
Energy per bit access	>1 pJ	2 pJ	100-1000 mJ	10 nJ	2 pJ	100 pJ	0.02 pJ
Leakage power	High	Medium	High	Low	Low	Low	Low
Endurance	>10^{15}	>10^{15}	>10^{15}	10^4	>10^{15}	10^5-10^9	10^5-10^{11}
Non volatlitly	No	No	Yes	Yes	Yes	Yes	Yes
Scalability	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Figure 2: Comparison of traditional memory and NVM technologies [39]

2.1 For compiler-oriented optimizations

We consider a compiler-based approach to mitigate the overhead of stores on STT-RAM-based cache. The role of the compiler is to present to the hardware a sequence of instructions that best exploits the underlying resources. Compilers, traditionally, are not directly exposed to energy aspects. However, with all the weight given to energy consumption, it is worthy to investigate how the compiler optimizations could have tremendous impacts on the overall energy consumption. As the main factor for activity on processor and memory systems, software has a signification effect on the energy consumption of the entire system. Still, developing new power-aware compiler optimizations and combine them with performance-oriented compiler optimizations is the focus of several researches.

As software execution corresponds to performing operations on hardware, as well as accessing and storing data, it requires power dissipation for computation, storage and communication. Beside the fact that the energy consumed during execution is very important, reducing the code size and thus, the storage cost of a program, is the classical goal of compilers. The energy cost of executing a program rely on the machine code and the target machine. Therefore, for a target architecture, energy cost is bound to machine code and consequently to compilation. Hence, it is the compilation process itself that strikes energy consumption.

Traditional optimizations on SRAM-based memories increase the performance of a program by reducing the work to be done during program execution [29, 1]. Memory optimizations play with latency and data placement. Optimizations such as loop tiling and register allocation try to keep data closer to the processor in order be accessed more faster. Memory optimizations contribute also in reducing energy consumption. Loop tiling tries to keep a value in an on-chip cache instead of an off-chip memory and register allocation optimization try to allocate data in a register instead of the cache. Both techniques allow to save power/energy due to reduced switching activities and switching capacitance [19].

2.2 Our contribution

After a preliminary evaluation of typical optimizations found in mainstream compilers such as gcc or clang, we propose a new dedicated compiler-based approach [7, 6, 8] to minimize energy consumption

on STT-RAM based cache by mitigating the drawback of such a technology. Our approach aim to reduce the number of write operations by reducing the number of silent stores, which are the stores that write a value that already exist in memory. In a previous study [31], we explored the feasibility of integrating STT-RAM at different cache hierarchy levels in ARM architectures. Both L1 instruction and data caches were evaluated with SRAM and STT-RAM technologies. Based on the promising gained insights, we now apply the silent store transformation in presence of STT-RAM in L1 cache.

2.3 Outline

The rest of this document is organized as follows. The related works are presented in Section 3. Background information about silent stores and the implementation of the optimization is provided in Section 4. In section 5, a set of experiments is conducted to evaluate the proposed approach. In Section 6, we provide important insights to a better outcome. Finally, Section 7 concludes this report.

3 Related works

We briefly discuss some existing work on the exploitation of STT-RAM in the memory architecture. The STT-RAM integration in the system is coupled with either hardware-level or compiler-oriented optimizations. This enables to mitigate a major limitation of STT-RAM w.r.t its costly write access.

3.1 Hardware-based optimizations/scheduling

There are several works that studied energy efficiency on NVMs-based hybrid cache using fully-hardware based mechanisms [37, 32, 35, 34]. Sun et al. in [37] proposed hybrid L2 cache consisting of MRAM and SRAM, and employed migration based policy to mitigate the drawbacks of MRAM. The idea is to keep as many write intensive data in the SRAM part as possible to reduce the number of write operations to the STT-RAM part. They designed an SRAM-MRAM hybrid L2 cache with 31 ways of MRAM and 1 way of SRAM. Considering the high probability that a program writes data to a specific group of memory blocks repeatedly, data on MRAM should be migrated to SRAM if they are frequently written. Authors in [35, 34, 32] proposed a methodology that combines the gem5 [5] and NVSim [15] simulators for assess the impact of STT-RAM integration at different cache levels. They explored the possible energy savings enabled by STT-RAM when integrated at both L1 and L2 cache levels.

Migration based techniques are widely recommended in recent works on hybrid caches [38, 18, 22]. Li et al. in [22] proposed a migration mechanism to handle read operations also. Memory blocks that are read frequently are migrated from SRAM into STT-RAM. Zhou et al. in [40] proposed a technique called Early Write Termination to significantly reduce write energy with no performance penalty. It is a write process with the capability of early termination in case of a redundant write. It is implemented at the circuit level and dos not require extra read to precede a write, hence, the implementation does not have any impact on performance.

Smullen et al. in [36] proposed an approach for redesigning STT-RAM memory cells to reduce the high dynamic energy and slow write latencies. They lower the non-volatility time and as a result of that, the write current is reduced. Chen et al. in [11] proposed a reconfigurable hybrid cache architecture , in which NVM is put in the last-level cache along with SRAM. They provide hardware-based mechanisms to dynamically reconfigure hybrid cache based on the cache demand. They introduced a power gating circuitry to allow powering on/off SRAM/NVM arrays at way level.

Hu et al. in [16] targeted embedded Chip Multiprocessors (CMPs) with ScratchPad Memory (SPM) and non-volatile main memory. They introduced data migration and recomputation techniques to reduce the number of write activities on non-volatile memories In data migration, data is stored temporarily on the SPM of other cores rather than written back to the main memory. In data recomputation, the number of write activities is reduced by discarding the data that should have been written back to the main memory and recomputing it when it is needed.

3.2 Compiler-based optimizations

To explore the advantages of the hybrid cache, migration-based techniques are commonly used to dynamically move/migrate write-intensive data from STT-RAM to SRAM. Yet, migrations require additional read and write operations for data movement and this overhead has an impact on the performance and the energy efficiency of STT-RAM based hybrid cache. Li et al. in [24] addressed this issue and proposed a compilation method called migration-aware code motion. This mechanism is conceived to change the data access patterns in memory blocks in order to minimize the overhead of migrations . It consists of scheduling instructions accessing the same memory block with the same operation close to each other.It is designed without any hardware modification. Li et al. in [23] also proposed a migration-aware compilation for STT-RAM based hybrid cache in embedded systems, by re-arranging data layout to reduce the number of migrations. They showed that the reduction of migration overheads improves energy efficiency and performance. They also presented in [25] a migration-aware cache locking approach to reduce the migrations by locking migration-intensive memory blocks into SRAM part of hybrid cache.

Li et al. in [26] targeted the problem of allocating program variables into hybrid SPM-based systems. They presented an ILP formulation exploring that variables that are not alive simultaneously can share the same memory address and a graph-coloring based polynomial-time algorithm to address the problem of hybrid SPM allocation.

Hu et al. in [17] presented an approach based on region partitioning to generate optimal data allocations for each region. The program is divided into regions and before starting the execution of each region, data management code is executed to generate a data allocation which is suitable for this region.

3.3 Current work

In this work, we consider a *compile-time* optimization consisting in eliminating redundant write accesses to non hybrid L1 cache memory designed with STT-RAM, while improving energy-efficiency. We adopt the so-called silent store elimination technique introduced by Lepak et al. [20], who presented two store squashing mechanisms at microarchitectural level, called realistic and perfect. The realistic mechanism consists in replacing all stores by a store verification, which is a load, a comparison and the store if it is not silent. The perfect mechanism follows a similar principle, while the verification is performed only for the stores that are already known to be silent. Here, we leverage the realistic mechanism at compiler level for portability and more flexibility contrarily to the hardware-oriented approaches found in literature

4 Silent stores optimization

4.1 Overview

A store is said to be *silent* if it writes a value to a memory address where the same value is already stored.

Silent-store have been initially proposed and studied by Lepak et al. in [21] where they suggested new techniques for aligning cache coherence protocols and microarchitectural store handling techniques to exploit the value locality of stores. Their studies showed that eliminating silent stores helps to improve uniprocessor speedup and reduce multiprocessor data bus traffic.

Previous work showed that there is a significant percentage of total silent stores. Bell et al. [2] outline 18 % to 64 % of total stores as silent for SPEC95 benchmarks. In our study, we choose to move the concept to the compiler level, and use a similar mechanism for silent stores detection and elimination to reduce the energy consumption of on STT-RAM-based architectures.

The initial proposal was devised in hardware, and different mechanisms for store squashing have been proposed. In this work, we propose a software approach which is good for portability, and requires no change to the hardware. Our approach consists of modifying the source code by inserting *silentness* verification before stores that are identified as silent (see Figure 3).

```
1   store @x = val
2   ...
```

Listing 1: Before transformation

```
1   load y = @val
2   cmp x, y
3   beq next
4   store @x, val
5 next:
6   ...
```

Listing 2: After transformation

Figure 3: Silent store transformation

The intuition behind this approach is to reduce the total number of writes by replacing silent stores by loads.The stores verification helps to check whether the identified stores are indeed silent during their execution. This implementation is a realistic implementation. An ideal implementation would be to execute one time the identified silent stores and avoid verifying them, assuming that the identified stores will be always silent. The verification process is composed of a load to store address, a comparison of the loaded value with the to-be stored value and a conditional branch. The store will eventually not be executed if the verification is satisfied. Not all stores are silent, and stores may not be silent all the time. Still, silent stores can be characterized to some extent. Our solution attempts to optimistically replace a store by a sequence of instructions that avoids it. Whenever the store happens to not be silent, it must be executed. This case incurs a penalty, which we want to minimize. The process of identifying the stores that are silent is explained later on.

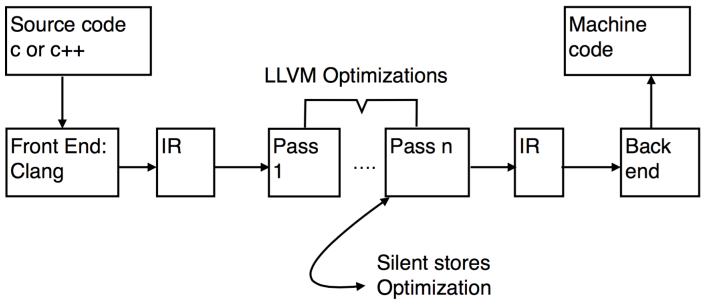

Figure 4: LLVM architecture

4.2 LLVM-based implementation

In order to apply the silent stores reduction, we implemented an LLVM-based optimization. The compilation framework is composed of two steps as follows:

- Step 1: Silent store profiling, based on memory profiling, collects information on all store operations to identify the silent ones;
- Step 2: Apply the optimization pass on the stores that are silent.

Our compilation process is a front-end framework. We focused on the intermediate representation (IR) as shown in Figure 4, where we implemented the silent stores optimization. While this figure describes a generic decomposition into basic steps, its instantiation in our case only contains two "Passes" (see above) between the IR boxes.

Through the first step, we want to get information about all the store instructions that are in a program. For that, we insert new instructions in the IR in front of every store to check whether the stored valued is equal to the already stored value. If it is the case, we increment a counter related to this particular store. Once the first step is done, we obtain a summary reporting how many times the store addresses have been accessed and how many times they have been silent. We also save their positions in the program so that we can identify them in the next step. Then, we move to the second step, where we apply the main part of the optimization taking into account the profitability threshold,. Hence, we apply the transformation only for stores that are likely to be more than 65% silent. We insert before every silent store that we identified from the previous step, the following instructions:

1. a load instruction to the store address;
2. a comparison instruction, to compare the written value to the already stored value;
3. a conditional branch instruction.

The objective is to execute the store instruction only if it is not silent as shown in Figure 3. The first step allows us to avoid inserting the checking instructions before all store instructions to minimize the possible overhead related to checks. The trad-off is between the cost of a write operation and the cost of a load, a compare and a conditional branch, in terms of time and energy. For each candidate

store, the transformation replaces the store by a sequence of load-compare-branch, and possibly a store (when not silent).

Since we're targeting ARM architectures, we take benefit from conditional instructions and particularly the conditional store **strne**. Therefore, the transformed store will be as showed in the Figure 5. Note that Thumb extension does not provide predication.

```
1   load y = @val
2   cmp x, y
3   storeNE @x, val
4 next:
5   ...
```

Figure 5: Transformation with conditional store

Using ARM ISA allows us to minimize the number of inserted instructions and therefore the impact of the transformation on performance: instead of inserting three instructions, we insert only two.

4.3 Motivational example

As a motivational example, we analyze the possible impact of memory on the energy consumed by an on-chip system. For the sake of simplicity, we consider a simple on-chip system model comprising a CPU, an L1 data denoted by L1d, an L1 instruction cache denoted by L1i, a bus and a memory controller. No other cache level (e.g. L2 or L3 cache) is taken into account, while the main memory is off-chip.

To design and evaluate this model, we use a number of simulation and power modeling tools already integrated in the MAGPIE automated design evaluation framework [12, 13, 33]. These tools include the gem5 [4] cycle-approximate simulator, combined with the NVSim [14] and McPAT [27] power, area and latency modeling tools, which respectively target non volatile and CMOS technologies. Here, the gem5 minor CPU model is used to define an ARM Cortex-A7 core running at 1 GHz, associated with L1i and L1d caches, both 32 KB and 4-way associative [9].

In order to assess the possible gain that one could expect from different memory technology candidates, we consider a simple program *P* illustrated in Fig. 6 (C and assembly codes). Since we are focusing on store instructions, we decided to carry out a preliminary experimental analysis on such a dedicated kernel with different execution scenarios. This simple code shows one store operation that is executed N times. We identify this store as a silent store.

Let us evaluate the cost of this simple code on a Cortex-A7 in terms of execution time, energy consumption and energy-delay product (EDP). In this work, EDP is considered as the main figure of merit for energy-efficiency as it combines both execution time and energy. By executing 50 million iterations of the simple program (i.e., $N = 50 \times 10^6$), we obtain the result reported in Table 1 and referred to as "full-SRAM". In this reference evaluation scenario, both L1d and L1i caches are in SRAM.

Now, by re-executing the same program while considering the L1 cache is entirely in STT-RAM instead of SRAM, we obtain the result displayed in Table 1, referred to as "full-STTRAM". Here, we

```
(original C code)


volatile int x = 0;
for (i=0; i<N;i++) {
  x=0; // silent
}
```

```
(corresponding assembly)


; i in r3, N in r4
.L3
  str  r2, [r1, #0]
  add r3, r3, #1 ; i++
  cmp r3, r4 ; i==N?
  bNE .L3
```

Figure 6: Dedicated kernel. The `volatile` keyword forces the compiler to keep the memory access on `x`, while other variables may be promoted to registers. The assignment in the loop repeatedly writes the same value 0 to the same memory location

Table 1: Evaluations of dedicated kernel: execution of original code with full SRAM L1 cache (ref.); execution of original and optimized codes with full STT-RAM L1 cache.

	Exec. time (ms)	Energy (mJ)	EDP
full-SRAM (ref)	305.13	79.5	24257.83
full-STTRAM	305.31 (+0.06 %)	67.8 (-14.7 %)	20700.01
full-STTRAM + silent store opt	305.31 (+0.06 %)	64.6 (-18.7 %)	19723.02

observe a reduction of the energy consumption thanks to the low static power inherent to the used non volatile memory. A very marginal increase is observed in the execution time, which is due to the penalty expected from higher write latency of STT-RAM compared to SRAM. Fortunately, this marginal increase is not important enough to cancel the gain in static power enabled by STT-RAM. As a result the EDP is improved.

Fig. 7 shows the application of the transformation on P. In Table 2, after transformation, we see how the number of instructions increases by approximately 100 million instructions, that correspond to 50 million load instructions and 50 million compare instructions. The number of cycles and the execution time did not increase, which means that the transformation on this small program didn't have an impact on performance. The key point is not to increase the total execution time, because leakage is proportional to time. Since the core represents a significant fraction of the total power, a slight increase is likely to offset the gain of the cache alone. Note that the reported results correspond to a scenario where the store is always silent, which means it's 100% silent.

Table 2: Some relevant metrics about the execution of P before and after silent stores transformation

Metric	Before transformation	After transformation
Execution time	305.31	305.31
Number of instructions	201 222 222	301 223 106
Number of cycles	305 304 382	305 312 970

```
1    .L3
2        str   r2, [r1, #0]
3        add r3, r3, #1
4        cmp r3, r4
5        bne .L3
```

Listing 3: Before transformation

```
1    .L3
2        ldr r0, [r1, #0]
3        cmp r0, r2
4        strne r2, [r1, #0]
5        add r3, r3, #1
6        cmp r3, r4
7        bne .L3
```

Listing 4: After transformation

Figure 7: Transformation on P (see Fig. 6)

4.4 Profitability Threshold

From the point of view of the data cache (considered in isolation), the optimization transforms a write into a read, possibly followed by a write. The write must occur when the store happens to not be silent. In the most profitable case, we replace a write by a read, which is beneficial due to the asymmetry of STT-RAM. On the contrary, a never-silent store results in write being replaced by a read and a write. Thus, the profitability threshold depends on the actual costs of memory accesses. In terms of energy cost we want:

$$\alpha_{read} + (1 - P_{silent}) \times \alpha_{write} \leq \alpha_{write}$$

where α_X denotes the cost of the operation X and P_{silent} is the probability of this store to be silent. This is equivalent to:

$$P_{silent} \geq \frac{\alpha_{read}}{\alpha_{write}}$$

Given the values of Table 3, we obtain a threshold of:

$$P_{silent} \geq 109/174 \approx 0.63, \text{i.e., a silentness percentage of } 63\,\%.$$

The above reasoning only considers cache memory accesses, and holds as long as the rest of computations does not heavily impact the system energy consumption compared to those cache memory accesses. We confirm it experimentally by considering the simple program shown in Fig. 8, where N represents the total number of store operations and M represents the number of silent instances/occurrence. We run the program with different values of M, so as to vary the silentness percentage. In Fig. 8, we observe, as expected, that higher probability of silentness reduces energy consumption of the L1 data cache. The silent store elimination is beneficial when the silentness percentage is above 63 %.

Table 3: 32 KB SRAM and STT-RAM memory estimation with NVSim

Technology	Latency		Dynamic Energy		Leakage
	Read (ns)	Write (ns)	Read (pJ)	Write (pJ)	(mW)
SRAM	1.31	1.19	24	6	44.70
STT-RAM	1.96	10.94	109	174	6.72

```
volatile int x;
for(i=0; i<N; i++)
{
     y=0;
     if (i>M)
          y=i;
     x=y;  // store
}
```

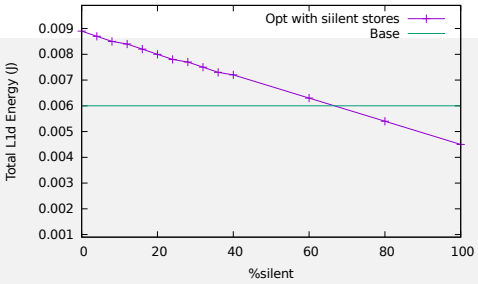

Figure 8: Determination of the profitability threshold of silent store elimination.

5 Experimental setup

5.1 Configuration

We consider a simple ARM monocore architecture based on Cortex-A7. We use an evaluation framework called MAGPIE [12], consisting of the gem5 simulator, combined with the NVSim and McPAT timing, power and area estimation tools (based on gem5 stats), see Figure 9. NVSim specifically target NVMs technologies. In our case, the STT-RAM technology is used. In a previous study [31], we explored the feasibility of integrating STT-RAM at different cache hierarchy levels in ARM architectures. Both L1 instruction and data caches were evaluated with SRAM and STT-RAM technologies. Based on the promising gained insights, we now apply the silent store transformation in presence of STT-RAM in L1 cache.

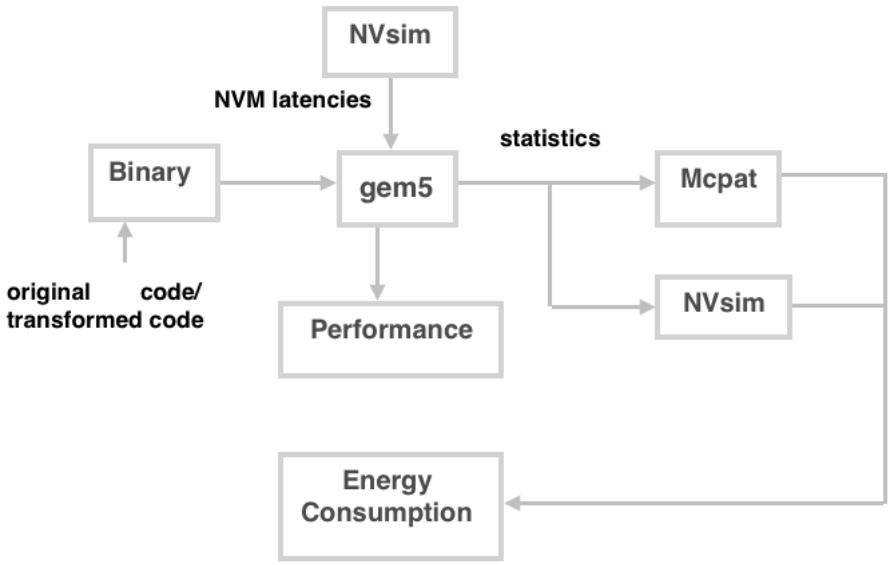

Figure 9: The simulation flow : MAGPIE

5.2 Experiments

In their seminal work [20], Lepak and Lipasti studied the SPEC95 benchmark suite in which high silentness percentages have been exposed. For instance, `vortex` and `m88ksim` reach respectively 64 % and 68 % of silent stores overall on PowerPC architecture (there was no per-store characterization in that paper). Such silentness levels could typically benefit from our optimization.

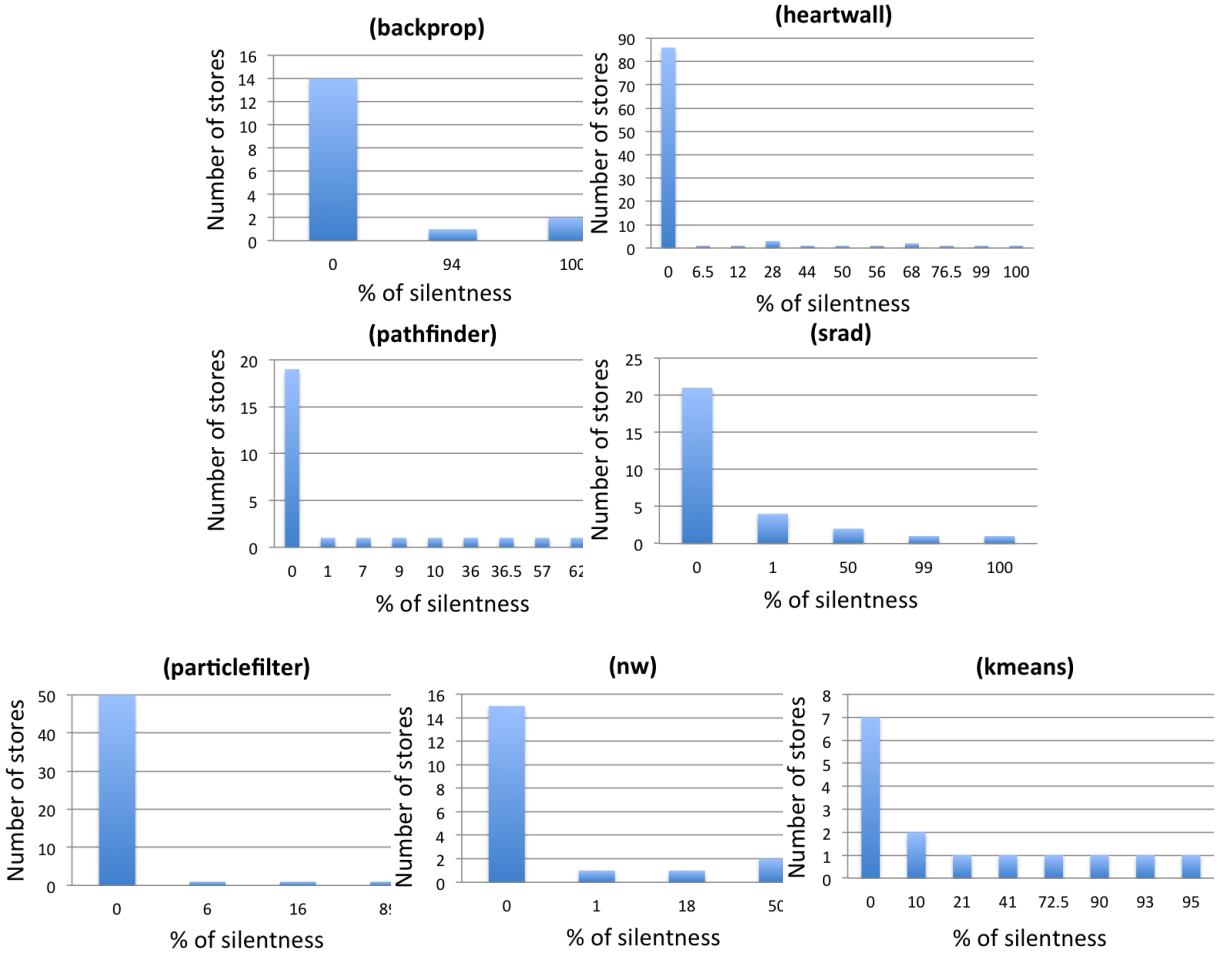

Figure 10: Analysis of silentness percentage for selected Rodinia applications.

More generally, the profitability threshold of our optimization for an application depends on the silentness distribution over its store instructions. Higher silentness percentages of these instructions favor more energy reduction. In order to explore this trade-off, let us consider a subset of applications defined in the Rodinia benchmark suite [10]: `backprop` , `heartwall`, `pathfinder`, `srad`, `particlefilter`, `nw` and `kmeans`. Fig. 10 summarizes a silentness analysis of their store instructions: they respectively contain 3 silent stores (i.e., silent at least once) out of 17 stores, 13 silent stores out of 99 stores, 9 silent stores out of 28 stores, 8 silent stores out of 29 stores, 3 silent stores out of 53 stores, 4 silent stores out of 18 stores and 8 silent stores out of 15 stores. The silentness distribution over these silent stores is reported in Fig. 10 via the displayed percentages. For instance, for `backprop` all 3 silent stores are highly silent: one at 94 % and two at 100 %, meaning that they

Table 4: Optimizing sample applications: SRAM and STTRAM configurations denote the original application program executed respectively with full SRAM and full STT-RAM L1 caches; STTRAM+opt denotes the code optimized for stores above the indicated silentness percentages and executed with full STT-RAM L1 caches.

App.	Config.	Exec. time (ms)	Energy (mJ)	EDP
backprop	SRAM	598.5	136.1	81455.85
	STTRAM	636.6	124.5	79256.70
	STTRAM+opt	621.7	122.1	75909.57
heartwall	SRAM	13399.6	3269.3	43807312.28
	STTRAM	13855.0	2773.0	38419915
	STTRAM+opt	13818.4	2764.1	38195439.44
pathfinder	SRAM	137.2	35.9	4925.48
	STTRAM	137.8	30.1	4147.78
	STTRAM+opt	137.6	30.1	4141.76
srad	SRAM	8154.8	1693.7	13811784.76
	STTRAM	8230	1479.50	12176285
	STTRAM+opt	8214.2	1476.8	12130730.56
particlefilter	SRAM	11255.83	2424.8	27293136.58
	STTRAM	11464.2	2133	24453138.6
	STTRAM+opt	11455.84	2132.10	24424996.46
nw	SRAM	440.3	100.8	44382.24
	STTRAM	455.73	88.6	40377.68
	STTRAM+opt	452.6	88.1	39874.06
kmeans	SRAM	118.11	28.1	3318.89
	STTRAM	126.5	24.9	3149.85
	STTRAM+opt	108.11	21.7	2345.99

can be transformed in order to improve the energy-efficiency with STT-RAM. This is visible in Table 4, where the STTRAM+opt scenario shows better results than STTRAM scenario, in terms of execution time and total energy consumption. The reported results in Table 4 corresponds to small inputs.

For the remaining applications, i.e., `heartwall`, `pathfinder`, `srad`, `particlefilter`, `nw` and `kmeans`, the silentness percentages of identified silent stores vary from 1 % to 100 %. Note that the maximum percentage for `nw` is 50%. Only the two silent stores corresponding to this percentage are optimized for profitability reason. All other applications are optimized by transforming their silent stores associated with at least 63% of silentness percentage. Table 4 summarizes the evaluation of all these optimizations. We carried out further investigations by evaluating the above applications while optimizing silent stores associated with at least an arbitrary silentness percentage of 15%. For most of the applications, no gain was observed except for `kmeans` and `pathfinder`. For these two applications, the gain is as good as with 63%. This suggests the profitability threshold of `kmeans` and `pathfinder` can be around 15%.

In order to understand how the optimization profitability threshold varies from an application to another, the number of store instances could be analyzed. For `kmeans`, this number associated with low silentness percentage, e.g. 15%, is negligible compared to the number of store instances associated with higher silentness percentages, e.g. 63%. Therefore, the overhead introduced by the optimization of stores with low silentness percentage is marginal enough to be mitigated globally. This explains the similar evaluation results observed with 15% and 63% thresholds for the two applications. However, in the other applications, the number of store instances associated with low silentness percentage is significant enough to induce a penalizing overhead resulting from our optimization of stores. Then, this translates into higher execution time and energy.

6 Analysis of optimization applicability

Executing instructions does not necessarily increase total execution time due to the synergy of compiler optimizations with several micro-architectural features of modern processors, as well as properties of applications. Many factors are involved in the impact of the presented optimization. The compiler itself and the architectural features play an important role. Compiler Scheduler orders instructions so that the resulting sequence minimizes the execution time. The compiler is free to swap instructions as long as they do not have any dependencies. The goal is then to maximize the utilization of hardware resources, yet avoid over-subscription. Scheduling is key for an in-order cores. Out-Of-Order cores are less sensitive since they have hardware mechanisms to generate a schedule at runtime, though they have a much more limited visibility on upcoming instructions.

```
1    .L3
2        str  r2, [r1, #0]
3        add  r3, r3, #1
4        cmp  r3, r4
5        bne  .L3
```

Listing 5: Before transformation

```
1    .L3
2        ldr  r0, [r1, #0]
3        cmp  r0, r2
4        beq  skip
5        str  r2, [r1, #0]
6        skip:
7        add  r3, r3, #1
8        cmp  r3, r4
9        bne  .L3
```

Listing 6: After transformation

Figure 11: Transformation without prediction P (see Fig. 6)

Architectural features have significant impact on compiler optimizations. In our case, since we led experiments on ARM cores, particularly cortex-A7, that supports ARM ISA as well as Thumb ISA, we found out that the difference between these two ISA can make an important change in the direction of this study. In fact, Thumb ISA does not support conditional instructions. Thus, *strne* will be automatically converted to a *branch+str*. For that, we studied the cost of the transformation with the branch as shown in the Figure 11. Then, we observed that the number of cycles increases significantly due to the branch which make us conclude that we take more benefit from this optimization by using ARM ISA where we have one fewer instruction, thus limiting the risk to increase execution time. As a side effect, it also lowers the number of branch mispredictions by eliminating the branch alltogether.

Hence, if we use this transformation under Thumb, the overhead of branch instructions must be concealed in order to see the optimization effect.

Moreover, superscalar processors can execute several independent instructions in parallel. For example, the ARM Cortex A7 is a dual-issue processor. In many cases, the processor is not able to fully exploit the available parallelism, leaving empty *slots*. When added instructions are able to fit in these unexploited slots, they do not increase execution time. Consider the example on Figure 12: in the original code, each instruction depends on its predecessor. Execution is purely sequential, even on a superscalalr processor. Our newly introduced instructions can be executed in the empty slots, as shown on the the right hand side of the figure. Note that the independent subtractions has been scheduled earlier to maximize overlap.

```
1   add r1=r2+r3
2   add r4=r1+2
3   mul r5=r4*r3
4   add r6=r5+3
5   store @x=r6
6   sub r4=r2-r3
7   sub r5=r4-1
8   ...
```

```
1   add r1=r2+r3 || load r0=@val
2   add r4=r1+2
3   mul r5=r4*r3
4   add r6=r5+3
5   sub r4=r2-r3 || cmp r6, r0
6   sub r5=r4-1  || beq next
7   store @x=r6
8 next:
9   ...
```

Figure 12: Example of superscalar execution. The original code (left) is purely sequential, the transformed code is able to take advantage of unexploited parallelism

Therefore, our silent stores elimination technique works best if we can hide the newly introduced instructions in empty slots. This requires superscalar capability. We rely on the compiler to schedule this instructions at the best possible locations in the original sequence of instructions. This is critical for in-order processors. OoO processors may be able to find an optimal sequence by themselves, but OoO also generates a denser schedule, with fewer empty slots. In order to assess the possible gain that one could expect from the transformation, the program should be analyzed before applying the silent stores transformation. If the instructions scheduling doesn't allow to hide the *silentness* verification, then the overhead of this verification will increase the execution time and therefore the total leakage.

7 Conclusion and perspectives

In this study, we presented how the implementation of a typical compiler optimization can contribute in mitigating the dynamic energy consumption issue related to NVMs at L1 cache level. This optimization eliminates silent stores in cache accesses, which in turn reduces the number of write transactions that are more expensive than read ones for NVMs, both in terms of latency and energy consumption. It has been fully implemented in LLVM and validated on a subset of Rodinia benchmark. Beyond the L1 cache illustrated in this work, this optimization is also beneficial to any memory hierarchy level integrating NVMs. From the previous analysis, a successful application of our compiler-level optimization requires a number of features for a better outcome. It provides better energy-efficiency results when the cost of newly introduced instructions can be hidden adequately. This requires superscalar capability for exploiting empty instruction slots. We also rely on the compiler to schedule these instructions at the most suitable locations in the original sequence of instructions. This is critical for in-order cores, while OoO cores are able to find an optimal sequence by themselves. The use of predicated instructions is another interesting feature that enables to mitigate the possible execution time overhead related by the extra instructions introduced by our optimization.

An extension of this work would be to automatize the program analysis phase for silent stores verification, thus, making the compilation process on three steps instead of two: detecting silent stores with the profitability threshold, analayze instructions schedulability and then apply the transformation except and only on the stores where the verification instructions will be covered by the parallelism of the execution.

References

[1] Aho, A. V., Sethi, R., and Ullman, J. D. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986. ISBN 0-201-10088-6.

[2] Bell, G. B., Lepak, K. M., and Lipasti, M. H. Characterization of silent stores. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 133–144. IEEE, 2000.

[3] Benini, L. and Micheli, G. d. System-level power optimization: Techniques and tools. *ACM Trans. Des. Autom. Electron. Syst.*, 5(2):115–192, April 2000. ISSN 1084-4309. doi: 10.1145/335043.335044. URL http://doi.acm.org/10.1145/335043.335044.

[4] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D., and Wood, D. A. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011. ISSN 0163-5964. doi: 10.1145/2024716.2024718. URL https://doi.org/10.1145/2024716.2024718.

[5] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D., and Wood, D. A. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011. ISSN 0163-5964. doi: 10.1145/2024716.2024718. URL https://doi.org/10.1145/2024716.2024718.

[6] Bouziane, R., Rohou, E., and Gamatié, A. How could compile-time program analysis help leveraging emerging nvm features? In *2017 First International Conference on Embedded Distributed Systems (EDiS)*, pages 1–6, 2017. doi: 10.1109/EDIS.2017.8284031.

[7] Bouziane, R., Rohou, E., and Gamatié, A. LLVM-based silent stores optimization to reduce energy consumption on STT-RAM cache memory. In *European LLVM Developers Meeting (EuroLLVM'17), Saarbrucken, Germany*, 2017.

[8] Bouziane, R., Rohou, E., and Gamatié, A. Compile-time silent-store elimination for energy efficiency: an analytic evaluation for non-volatile cache memory. In Chillet, D., editor, *To appear in the proceedings of the RAPIDO'18 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, Manchester, UK*. ACM, 2018.

[9] Butko, A., Bruguier, F., Gamatié, A., Sassatelli, G., Novo, D., Torres, L., and Robert, M. Full-system simulation of big.little multicore architecture for performance and energy exploration. In *10th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip, MCSOC 2016, Lyon, France, September 21-23, 2016*, pages 201–208. IEEE Computer Society, 2016. doi: 10.1109/MCSoC.2016.20. URL https://doi.org/10.1109/MCSoC.2016.20.

[10] Che, S., Sheaffer, J. W., Boyer, M., Szafaryn, L. G., Wang, L., and Skadron, K. A characterization of the rodinia benchmark suite with comparison to contemporary CMP workloads. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'10)*, 2010. ISBN 978-1-4244-9297-8.

[11] Chen, Y.-T., Cong, J., Huang, H., Liu, B., Liu, C., Potkonjak, M., and Reinman, G. Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, 2012.

[12] Delobelle, T., Péneau, P., Gamatié, A., Bruguier, F., Senni, S., Sassatelli, G., and Torres, L. MAGPIE: System-level Evaluation of Manycore Systems with Emerging Memory Technologies. In *Workshop on Emerging Memory Solutions - Technology, Manufacturing, Architectures, Design and Test at Design Automation and Test in Europe - DATE'2017, Lausanne, Switzerland*, 2017.

[13] Delobelle, T., Péneau, P.-Y., Senni, S., Bruguier, F., Gamatié, A., Sassatelli, G., and Torres, L. Flot automatique d'évaluation pour l'exploration d'architectures à base de mémoires non volatiles. In *Conférence d'informatique en Parallélisme, Architecture et Système, Compas'16, Lorient, France*, 2016.

[14] Dong, X., Xu, C., Xie, Y., and Jouppi, N. P. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(7):994–1007, 2012. doi: 10.1109/TCAD.2012.2185930.

[15] Dong, X., Xu, C., Xie, Y., and Jouppi, N. P. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(7):994–1007, 2012. doi: 10.1109/TCAD.2012.2185930.

[16] Hu, J., Xue, C. J., Tseng, W.-C., He, Y., Qiu, M., and Sha, E. H.-M. Reducing write activities on non-volatile memories in embedded cmps via data migration and recomputation. In *Proceedings of the 47th Design Automation Conference*, DAC '10, 2010.

[17] Hu, J., Xue, C. J., Zhuge, Q., Tseng, W., and Sha, E. H. Data allocation optimization for hybrid scratch pad memory with SRAM and nonvolatile memory. *IEEE Trans. VLSI Syst.*, 2013.

[18] Jadidi, A., Arjomand, M., and Sarbazi-Azad, H. High-endurance and performance-efficient design of hybrid cache architecture through adaptive line replacement. In *Proceedings of the 2011 International Symposium on Low Power Electronics and Design, 2011, Fukuoka, Japan, August 1-3, 2011*, pages 79–84, 2011.

[19] Kremer, U. Low power/energy compiler optimizations, 2004.

[20] Lepak, K. M. and Lipasti, M. H. On the value locality of store instructions. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA '00, 2000.

[21] Lepak, K. M., Bell, G. B., and Lipasti, M. H. Silent stores and store value locality. *IEEE Transactions on Computers*, 50(11):1174–1190, 2001.

[22] Li, J., Xue, C. J., and Xu, Y. STT-RAM based energy-efficiency hybrid cache for cmps. In *IEEE/IFIP 19th International Conference on VLSI and System-on-Chip, VLSI-SoC 2011, Kowloon, Hong Kong, China, October 3-5, 2011*, 2011.

[23] Li, Q., Li, J., Shi, L., Xue, C. J., and He, Y. Mac: Migration-aware compilation for stt-ram based hybrid cache in embedded systems. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, 2012.

[24] Li, Q., Shi, L., Li, J., Xue, C. J., and He, Y. Code motion for migration minimization in stt-ram based hybrid cache. In *Proceedings of the 2012 IEEE Computer Society Annual Symposium on VLSI*, ISVLSI '12, 2012.

[25] Li, Q., Zhao, M., Xue, C. J., and He, Y. Compiler-assisted preferred caching for embedded systems with stt-ram based hybrid cache. In *Proceedings of the 13th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, Tools and Theory for Embedded Systems*, LCTES '12, 2012.

[26] Li, Q., Zhao, Y., Hu, J., Xue, C. J., Sha, E., and He, Y. Mgc: Multiple graph-coloring for non-volatile memory based hybrid scratchpad memory. *2012 16th Workshop on Interaction between Compilers and Computer Architectures (INTERACT)*, 2012.

[27] Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., and Jouppi, N. P. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, page 469–480, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605587981. doi: 10.1145/1669112.1669172. URL https://doi.org/10.1145/1669112.1669172.

[28] Mittal, S. and Vetter, J. S. A survey of software techniques for using non-volatile memories for storage and main memory systems. *IEEE Trans. Parallel Distrib. Syst.*, 27(5):1537–1550, 2016. doi: 10.1109/TPDS.2015.2442980. URL http://dx.doi.org/10.1109/TPDS.2015.2442980.

[29] Muchnick, S. S. *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. ISBN 1-55860-320-4.

[30] Panda, P. R., Dutt, N. D., Nicolau, A., Catthoor, F., Vandecappelle, A., Brockmeyer, E., Kulkarni, C., and de Greef, E. Data memory organization and optimizations in application-specific systems. *IEEE Design & Test of Computers*, 18(3):56–68, 2001. doi: 10.1109/54.922803. URL http://dx.doi.org/10.1109/54.922803.

[31] Péneau, P., Bouziane, R., Gamatié, A., Rohou, E., Bruguier, F., Sassatelli, G., Torres, L., and Senni, S. Loop optimization in presence of STT-MRAM caches: A study of performance-energy tradeoffs. In *26th International Workshop on Power and Timing Modeling, Optimization and Simulation, PATMOS 2016, Bremen, Germany, September 21-23, 2016*, pages 162–169, 2016. doi: 10.1109/PATMOS.2016.7833682. URL http://dx.doi.org/10.1109/PATMOS.2016.7833682.

[32] Senni, S., Torres, L., Sassatelli, G., Gamatié, A., and Mussard, B. Emerging non-volatile memory technologies exploration flow for processor architecture. In *2015 IEEE Computer Society Annual Symposium on VLSI*, pages 460–460, 2015.

[33] Senni, S., Delobelle, T., Coi, O., Péneau, P., Torres, L., Gamatié, A., Benoit, P., and Sassatelli, G. Embedded systems to high performance computing using stt-mram. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 536–541, 2017. doi: 10.23919/DATE.2017.7927046.

[34] Senni, S., Torres, L., Sassatelli, G., Gamatié, A., and Mussard, B. Emerging non-volatile memory technologies exploration flow for processor architecture. In *2015 IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2015, Montpellier, France, July 8-10, 2015*, page 460. IEEE Computer Society, 2015. doi: 10.1109/ISVLSI.2015.126. URL https://doi.org/10.1109/ISVLSI.2015.126.

[35] Senni, S., Torres, L., Sassatelli, G., Gamatié, A., and Mussard, B. Exploring MRAM technologies for energy efficient systems-on-chip. *IEEE J. Emerg. Sel. Topics Circuits Syst.*, 6(3):279–292, 2016. doi: 10.1109/JETCAS.2016.2547680. URL https://doi.org/10.1109/JETCAS.2016.2547680.

[36] Smullen, C. W., Mohan, V., Nigam, A., Gurumurthi, S., and Stan, M. R. Relaxing non-volatility for fast and energy-efficient stt-ram caches. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, HPCA '11, 2011.

[37] Sun, G., Dong, X., Xie, Y., Li, J., and Chen, Y. A novel architecture of the 3d stacked MRAM L2 cache for cmps. In *15th International Conference on High-Performance Computer Architecture (HPCA-15 2009), 14-18 February 2009, Raleigh, North Carolina, USA*, 2009.

[38] Wu, X., Li, J., Zhang, L., Speight, E., Rajamony, R., and Xie, Y. Hybrid cache architecture with disparate memory technologies. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, 2009.

[39] Zhao, J., Xu, C., Chi, P., and Xie, Y. Memory and storage system design with nonvolatile memory technologies. *IPSJ Trans. System LSI Design Methodology*, 8:2–11, 2015. doi: 10.2197/ipsjtsldm.8.2. URL http://dx.doi.org/10.2197/ipsjtsldm.8.2.

[40] Zhou, P., Zhao, B., Yang, J., and Zhang, Y. Energy reduction for STT-RAM using early write termination. In *2009 International Conference on Computer-Aided Design, ICCAD 2009, San Jose, CA, USA, November 2-5, 2009*, 2009.