ARCHITECTING EMERGING MEMORY TECHNOLOGIES FOR ENERGY-EFFICIENT COMPUTING IN MODERN PROCESSORS

WANG JIANXING

(B.Sc., University of Macau)

THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SINGAPORE
2015

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety.

I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Wang Jx

Wang Jianxing
Nov 26, 2015

Acknowledgements

The journey towards PhD is never easy. It can be long and tough, and makes you frustrated because of the unknown nature of academic research. Fortunately, I have come so far and are now near the end of this journey. Along the path it has taught me a lot of things, including the knowledge of a particular area, the ability of critical thinking, enhancing/realizing an idea, and reading/writing research papers. More importantly, I have learned the way of getting along well with the world, and the way of living well with yourself.

Needless to say, I am not able to achieve this without the help from a bunch of great people. First of all, I shall thank NUS and SOC to offer me such a position with great environment for doing academic research. The opportunities of getting into a world-class university are always limited and I am so lucky to have one. Secondly, I shall thank ICA for processing my student visa and MOE for providing me tuition grant and scholarship. In addition, when my scholarship ended, I shall thank MOM for approving my job application so that I can continue being supported financially.

Next, I shall thank the administrative/service staffs from Registrar's office, Dean's office and Technical services for their swift processing of various kinds of document and service. Then I shall thank the wonderful lecturers from both

SOC and CELC for their excellent teachings, including but not limited to: Dr.Lee Ming Cherk, Mrs.Brenda Lee, Dr.Tulika Mitra, Dr.Sung Wing Kin (Ken), Dr.Soo Yuen Jien and Dr.Wong Weng-Fai. Special thanks goes to Ken, who was my first supervisor and helped me a lot through the very early stage of my PhD.

Research-wise, I shall thank my thesis committee for examining my GRP, TP and final thesis. I shall also thank the people who have given me constructive comments, including all collaborators and paper reviewers. These people include but not limited to: Dr.Soo Yuen Jien, Dr.Teo Yong Meng, Dr.Li Hai (Helen), Dr.Chen Yiran and my supervisor Dr.Wong Weng-Fai. Special thanks goes to Helen, who helped me a lot in terms of research ideas and background knowledge. I shall also thank my research group who were there on time for every Friday's meeting and we had lots of chat on both academics and lives. They include: Dr.Wang Chundong, Mr.Manoharan and Dr.Pooja Roy.

Life-wise, I shall thank a dozen of friends for lighting my life in Singapore with happiness and joy. Although many of them have left or will soon leave this tiny place for a better life, I still think of them once in a while and imagine how boring and lonely my PhD life will be without them. The list includes but not limited to: Wang Fangda, Chen Tao, Liu Shuang, Ye Nan, Cao Nannan, Lim Zhanwei, Li Jiekun, Jiao Huipeng, Wang Jingjing, Zuo Zhiqiang, Luo Chengwen, Li Zhuolun, Zhai Jing and Wu Kegui. I wish them all the best in their future endeavors.

As usual, special thanks goes to my family. They unconditionally provided me all kinds of support I need during this five years. Without them I will never achieve so far. Also thank my girlfriend for the morale support given when I felt down and upset.

Finally, I sincerely thank my supervisor Dr.Wong Weng-Fai. I believe he is the top-3 nicest professor in the department. He is keen, kind and knowledgeable. He trust his students and always put faiths on them. Sometimes I felt he is like a father to me rather than a supervisor. In particular, his contribution includes but not limited to

- Support me financially with scholarship, RAship and conference travel grant.
- Purchase various hardwares and softwares as necessary for our research work.
- Give numerous advices, comments, suggestions, ideas and insights about my research.
- Occasionally treat us for meals.

In conclusion, this thesis will not exist without his guidance and generous support in various aspects.

Table of contents

Abstract

The concern about energy consumption in current technology node is becoming more serious. As we are already in deep sub-micron era, the relentless process scaling makes existing energy reduction techniques less efficient. It is predicted that we might soon face the situation where most of the silicon area is forced to be *dark* and can not be powered on simultaneously.

As alternative solutions, a number of *emerging memory* technologies are being actively studied. They allow us to design more energy efficient architectures and potentially solve the dark silicon problem from the ground. However, they also incur certain weaknesses such as high write energy and limited endurance. Previous studies showed that an effective way to mitigate these negative impacts is by optimizing the memory architecture.

Therefore, this thesis will focus on architecture designs for deploying emerging memories at various levels in the memory hierarchy. First of all, we exploit both the storage and computing capabilities of a particular emerging memory: the *memristor*, in the design of a neural branch predictor. Experiments showed that our design not only increases the accuracy, but also consumes less energy than traditional schemes. Secondly, we present a L1 cache architecture that utilizes both the conventional *SRAM* and the emerging *STT-RAM* technology.

Our scheme mitigated the performance impacts from the expensive write operations of STT-RAM and achieved energy savings with enhanced reliability. Thirdly, we propose a dynamic block reconfiguration mechanism in the design of multi level cell (MLC) STT-RAM last-level caches. Our design leveraged the latency/capacity trade-off in MLC STT-RAM to accelerate some portions of the cache with less energy consumption. Lastly, the use of STT-RAM in GPGPU has also been investigated. A hybrid L2 architecture employing a large STT-RAM part and a small SRAM augment is demonstrated. Our design monitors the read/write behavior of cache blocks in STT-RAM part and attempts to offload writes to SRAM augment, reducing the write penalties. Simulation shows our design achieved minor performance improvement over both pure SRAM and STT-RAM with significant energy reduction.

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Challenges of CMOS Scaling

The semiconductor industry has been fundamentally driven by *Moore's law* thus far, which states the number of transistors in integrated circuits will *double* every 18 months [61]. As shown in Figure 1.1, the number of transistors in modern microprocessors grows exponentially from few thousands (Intel 8080) during the last 70s to over 2 billion in 2011 (Intel Xeon E5). In the mean time, the size of transistors has been reduced over two dozen of generations from few micrometers to merely tens of nanometers.

One of the important rule to maintain the speed of transistor shrinking is *Dennard's scaling* [24]. It states that the supply and threshold voltages should scale proportionally with the technology node, and hence the power density will remain as a constant from one generation to another. While the CMOS process continues to scale, Dennard's scaling is no longer held due to the process variability under nanometer scale. At the same time, a number of challenges have emerged such

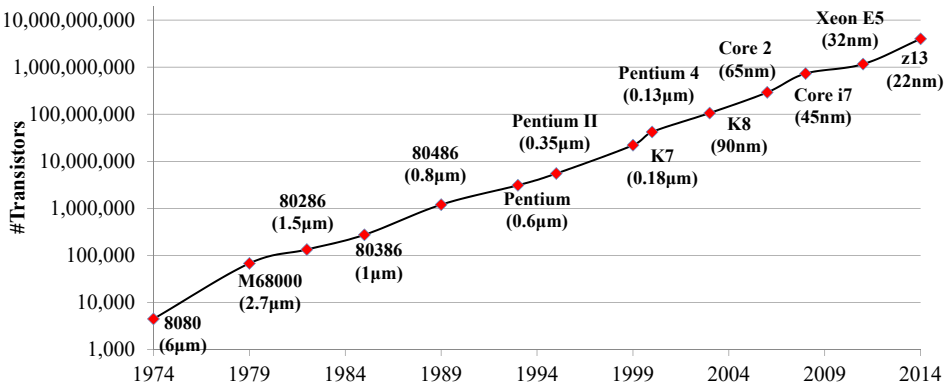


Fig. 1.1 Transistor counts in modern microprocessors.

as the RC delay problem [6], power dissipation [56], and the memory bandwidth wall [71]. In particular, the power consumption per chip area is trending upwards and circuit-level techniques can hardly mitigate such a trend without impacting performance. This is already happening in the current technology node, and is expected to get worse as further scaling will pack more transistors onto the silicon die, leading to unrealistic power consumption and heat density that cannot be handled by a conventional cooling system.

1.1.1 Thermal Design Power and Dark Silicon

Thermal design power (TDP) is the maximum amount of power that can be supplied to the chip so that the amount of heat generated during normal operation can be dissipated by the cooling system. While the density of MOSFETs continues to scale, the fixed chip-level TDP seriously constrain the active portion of the chip that can be powered on simultaneously at full frequency. Such a portion of the chip area that must be powered off or under-clocked so as not to exceed the given energy budget and cooling capability is called the *dark silicon*. Figure 1.2

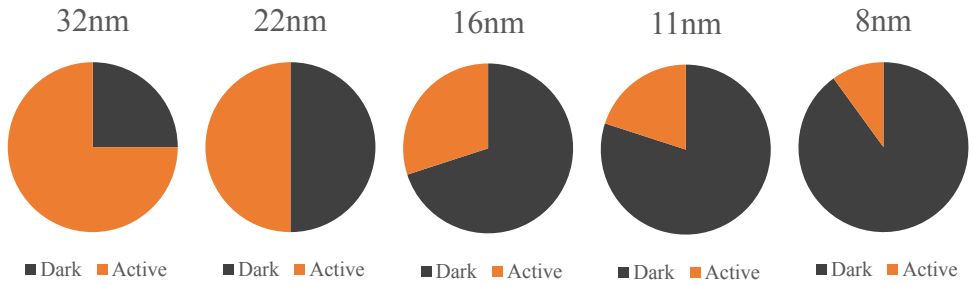


Fig. 1.2 Fractions of dark silicon projected for future process generation.

displays the fractions of dark silicon on-chip projected for future process generations based on previous studies [29, 74]. Theoretical data from *International Technology Roadmap for Semiconductors* (ITRS) shows that merely after two generations, half amount of the transistors on-chip can not be activated fully for performance due to the TDP constrain. Eventually, we will reach the point where most of the chip area is forced to go dark and deemed under-utilized.

1.1.2 Issues of On-chip Memory/Cache

One of the main battlefields where researchers are actively seeking solutions to turn the dark silicons back to performance is the on-chip memory/cache. Traditionally, on-chip memory is constructed with *static random access memory* (SRAM) because of its low latency, high endurance and scalability. However, a typical single bit SRAM cell is made of 6 - 10 transistors and leaves a large silicon footprint (20 - 50%) to the chip area [71]. Figure 1.3 shows the die of a nehalem-based processor from Intel where the SRAM cache area is roughly equal to the area of three cores. Furthermore, since SRAM uses bistable flip-flops to store data bits, it needs a constant supply of power to reinforce the cell for keeping the data intact which consumes a high amount of static/leakage energy.

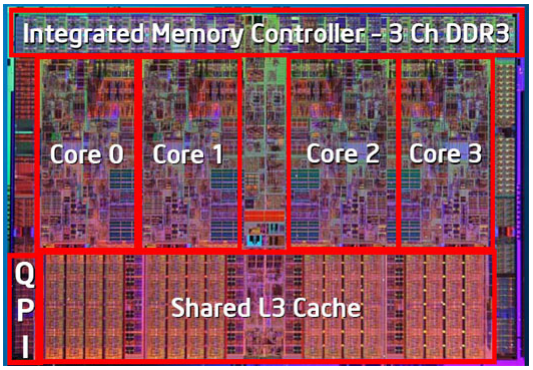


Fig. 1.3 The die of a Nehalem-based quad-core processor from Intel [3].

As the current design trend shifts to multi-core and many-core [20] system, the demands of larger and yet faster on-chip memory such as cache are growing. In order to mitigate the high static power of SRAM caches, a number of architectural designs have been proposed including *drowsy cache* [31], *cache decay* [47], and *filter cache* [48]. These schemes are good in their own ways, but they does not solve the fundamental problem and are becoming less efficient as the relentless process scaling continues. The static energy consumed by SRAM memories are seriously challenging the future design of high-performance and yet energy-efficient processor architectures.

1.2 Emerging Memory Technologies

To address the fundamental issue of SRAM's static/leakage power dissipation, some researchers are exploring a number of emerging memory technologies as potential augments/replacements. Examples include *magneto-resistive RAM* (MRAM) [27], *resistive RAM* (ReRAM) [95], *Domain wall memory* (DWM) [7] and *embedded DRAM* (eDRAM) [22]. These memories have several promising

features, such as high density (small feature size), non-volatility and CMOS-compatibility. When used in designing on-chip caches, they can be made in much larger capacities on the same chip area as SRAM, to satisfy the requirements of many-core systems and data-intensive applications. More importantly, their non-volatilities means a near-zero static/leakage energy consumption, which eliminate the weakness of standard SRAM.

As these emerging memory technologies are maturing from lab to manufacturing [4], system-level designers have started to design new memory architectures [59]. Since these new memory technologies differ from the conventional ones in a range of aspects including performance, energy and reliability, the existing architectures have to be optimized for them. Overall, they provides us a path towards a more energy-efficient computing environment.

1.3 Motivation and Goal

As these emerging memory technologies are becoming more and more attractive for future memory hierarchies, it is important to understand both the benefits they can bring and the limitations/downsides involved. In this dissertation, we would like to explore the possibilities in architecting these emerging memories at various levels of the memory hierarchy. Specifically, several architecture designs will be proposed for those technologies, in order to maximize their benefits while mitigate the penalties. Here we summarize the scope and attempts of this thesis in a comprehensive way -

- For different levels of the memory hierarchy, we shall choose an appropriate type of emerging memory that shares similar characteristics to the

existing technology deployed at the same level. In addition, such a choice shall consider the pros and cons of the technology together with the data access behavior at that level of the memory hierarchy.

- Secondly, the deployment of emerging memories is not a one-for-one replacement for most of the cases. They exhibit certain drawbacks that would incur performance and energy overheads. Therefore, existing memory architectures have to be modified in order to reduce the potential penalties while enjoying the benefits they bring.

- Lastly, the new architectures shall be properly evaluated. The experiments shall cover both the positive and negative aspects of the technology and the overall system performance including any overheads involved within the designs.

Overall, we attempt to answer the following question -

How to optimize architecture-level designs for the emerging memories in order to maximize their performance and energy benefits while alleviating their drawbacks?

1.3.1 Leveraging Emerging Memories in Architecture Design

Traditional memory hierarchy design usually consists of a number of levels: on-chip registers/buffers/caches implemented in SRAM, off-chip primary memory implemented in DRAM and persistent storage implemented in magnetic hard disks (HDD) or NAND-flash-based solid state drives (SSD). As the memory goes closer to the processor, performance becomes more critical so the requirements of latency/bandwidth are more strict. At the same time, it comes with

capacity penalty and shall be compensated by some larger-but-slower lower level memories.

However, as technology continues to scale down, the increasing power consumption and heat density have seriously constrained the future memory hierarchy designs. These challenges are posed by fundamental technology limits, making the existing device and architecture techniques less effective. On the other hand, as emerging memory technologies mature, integrating them into the current memory hierarchy has become an attractive choice. One way of doing this is to create a *hybrid* memory system where more than one kind of technologies are deployed at a level of the memory hierarchy. Such a design methodology combines the advantages from different technologies while mitigating their drawbacks.

In this thesis, we follow such an idea in designing new memory architectures with focus on energy efficiency. At the same time, we try to tackle the possibility of using emerging memory as both a computation and a storage component.

1.4 Overview and Contribution of the Thesis

Figure 1.4 illustrates the scope of this thesis and how it influences the pipeline design. A typical out-of-order pipeline includes instruction fetch/decode, registers renaming, scheduling, execution/memory operations and result write back/commit stages. To better unitize the resources, branch predicator has been introduced to enable speculative execution and the instruction/data caches are usually backed by a larger last-level cache.

First of all, as a key pipeline component, *branch predictor* can seriously affect single-thread performance. Recently, people introduced neural-based methods

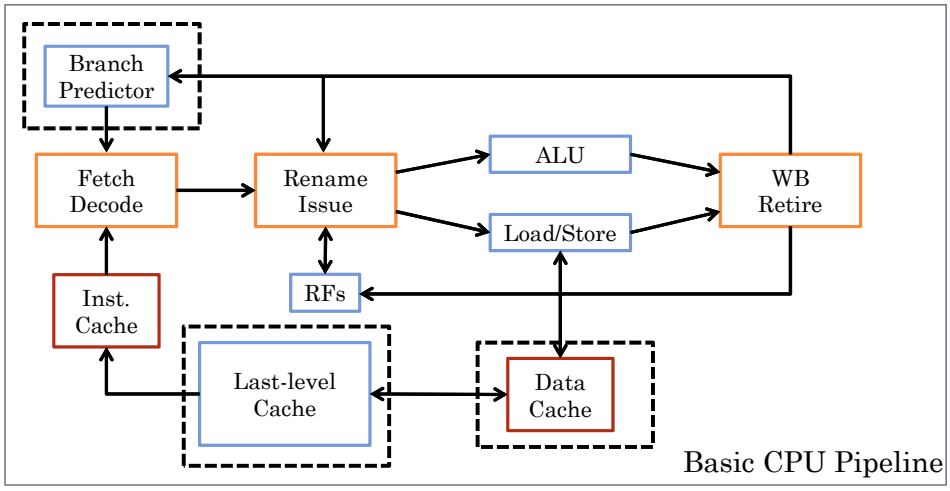


Fig. 1.4 Overview of the scope of the thesis (CPU).

into branch prediction [41]. These predictors are more resource-efficient and more accurate than the traditional method such as *gshare* [55]. However, they inhibit a relatively long prediction latency which prevent a wider adoption in commercial processors. Interestingly, because of their neural nature, the long prediction latency can be easily mitigated by using analog circuit techniques. Therefore, we propose an analog neural predictor based on the newly discovered *memristor* device. Memristor is a fundamental electrical component that can be used in analog computing. Both circuit and architecture design will be proposed and evaluated. The target is to construct a more practical neural predictor using emerging memory with high accuracy and less energy consumption. Chapter 3 will elaborate more on this work.

Secondly, we propose to deploy another promising emerging memory technology, *STT-RAM* in L1 cache. Since L1 cache is at the highest level of memory hierarchy, performance and reliability requirements are more critical than the lower level memory. Therefore, the issue of long write latency and relatively low

endurance of STT-RAM must be carefully managed. Observed that the read and write operations are usually distributed unbalanced across different cache blocks in L1, using a small SRAM partition to filter out the writes could enhance the overall performance and reduce lifetime failures of STT-RAM. Thus, the idea of *hybrid cache* is presented with the help of the cache coherent protocol. We leverage the built-in coherence information to predict which partition a block should stay in order to minimize performance penalty and energy consumption. In addition, block migration is introduced and two transfer policies are discussed and evaluated. This work is presented in Chapter 4.

Then, chapter 5 describes a scheme of using STT-RAM in *last-level cache* (LLC). While the hybrid L1 cache used the more mature single-level cell (SLC) design of STT-RAM , this work tries to architect multi-level cell (MLC) STT-RAM in LLC design. The main challenge of MLC comes from the two-step read/write scheme [9]. In order to differentiate the two data bits stored in a MLC, a two-stage data sensing is required. Similarly, write operation demands for two-step programming. Although using MLC STT-RAM can theoretically achieve $2\times$ data density, such a prolong access/programming latency incurs additional overheads and degrade overall performance. However, by nature such an enlarged capacity is not always necessary. Certain applications are more latency-sensitive so thus benefit more from lower access timings. For MLC STT-RAM, if we discard the *hard-bit*, the access to the *soft-bit* can be accelerated since only single-step sensing/programming is sufficient. Thus, we propose an optimization scheme for a MLC-based STT-RAM LLC which dynamically changes the block size to trade capacity for performance or vice-verse during application's runtime.

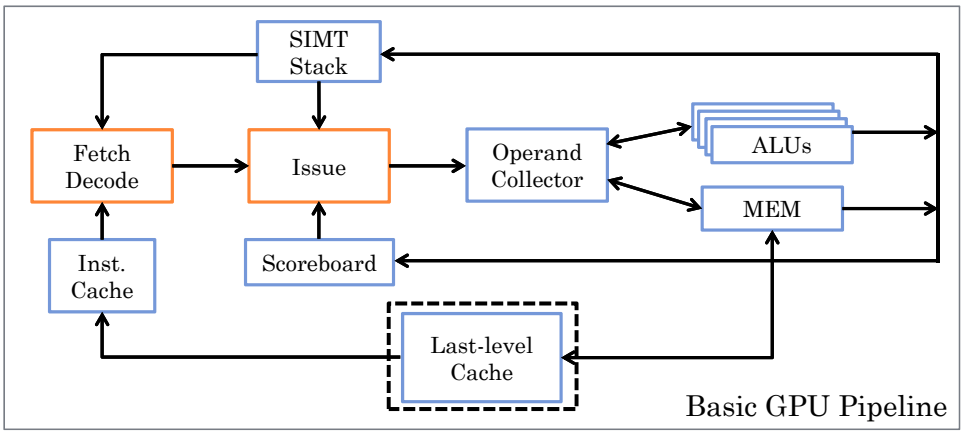


Fig. 1.5 Overview of the scope of the thesis (GPU).

Later in chapter 6, we move the focus to GPGPU architecture. Figure 1.5 shows a typical GPGPU pipeline that include fetch/decode, instruction issue, operand collection, ALU and memory stages. As GPGPU architectures are improving rapidly, the trend of larger L2 caches leads to increased energy consumption. While emerging STT-RAM has shown lots of desired properties in replacing CPU caches, its deployment in GPGPU cache is also promising. Because of the differences in architecture and application behaviors between CPU and GPU, new design methodology is required for STT-RAM to be deployed in GPU. Thus, we propose a hybrid L2 cache composed of a large STT-RAM part and a small SRAM augment. Together with differential block allocation and block migration scheme, the design managed to offload large portion of writes to SRAM. Both performance and energy evaluations will be presented in the chapter.

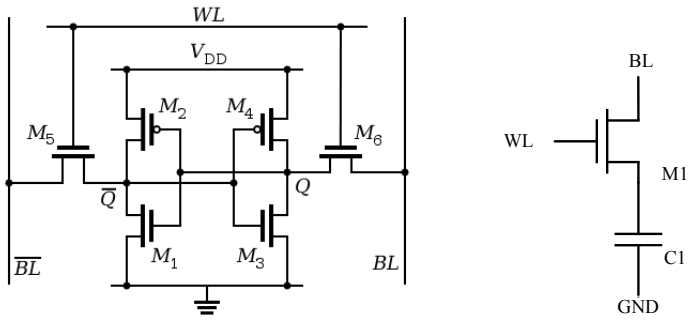
Finally in last chapter, we summarize the works have been done so far and point out future research direction after this thesis.

Chapter 2

Background

This chapter serves as an overview of both mainstream and emerging computer memory technologies. Traditionally, we classify *electrically addressed memories* into two categories: *volatile* and *non-volatile*. Volatile memories refer to those storage devices that require a continuous power supply for maintaining the data, while non-volatile memories can retain the contents without power supply. In the conventional domain, *static random-addressable memory* (SRAM) and *dynamic random-addressable memory* (DRAM) are two representatives in the volatile category while *read-only memory* (ROM) and *flash memory* are considered to be non-volatile.

Most of the emerging memories belong to the non-volatile group, including *ferroelectric* RAM (FeRAM), *phase-change memory* RAM (PCM), *resistive* RAM (ReRAM), *magneto-resistive* (MRAM) and *domain-wall memory* (DWM). *Embedded* DRAM (eDRAM) is an emerging volatile memory that works similarly to DRAM. The following sections will first describe the basic working principles of four conventional memories followed by six emerging memories.



(a) 6T SRAM (b) 1T-1C DRAM

Fig. 2.1 The cell structures of SRAM and DRAM.

In particular, *memristor* and *STT-RAM* will be described in great details since this thesis is mainly based on them.

2.1 Conventional Memories

2.1.1 Static RAM

Static RAM (SRAM) is a basic type of semiconductor memory that stores data bit in bistable latching circuitry. There are several types of SRAM with different number of transistors, but the most widely used one is 6-transistor (6T) type. A typical 6T SRAM design is shown in Figure 2.1a. Here two cross-coupled inverters are constructed with four *metal-oxide-semiconductor field-effect transistor* (MOSFETs) (M1 to M4) and two access transistors M5 and M6. When the word-line (WL) is turned on, depending on the value of Q, only one pair of MOSFETs (M1/M4 and M2/M3) will be activated. To perform a read operation, the bit-lines (BL and BL') will be pre-charged first and then be pulled down by Q and Q'. For write operation, BL and BL' are driven to some voltages so that they can overpower the values of Q and Q'.

When WL is not asserted, M5 and M6 are both turned off and the cell stays in *standby* state. A continuous power supply is required for holding the state of Q and Q' as the two cross-coupled inverters are enforcing each others. Otherwise, the data content would eventually lose. The primary advantage of SRAM over other memories is performance. Compared to DRAM, the read/write operation on flip-flops is significantly faster. However, because of the complexity, SRAM takes lot more space of the chip. Furthermore, reinforcement of the two cross-coupled inverters requires a constant current flow in M1 to M4, consuming energy when the cell in standby state.

2.1.2 Dynamic RAM

Dynamic RAM (DRAM) stores information using a fundamental electrical element: the *capacitor*. Capacitors can store energy in the form of electric charges. A typical 1-transistor 1-capacitor (1T-1C) DRAM cell is shown in Figure 2.1b. Transistor M1 controls the access to capacitor C1. The cell discharges during a read operation and is charged to an appropriate value in a write operation. While reading the content will also destroy the charge inside the capacitor, rewritten is necessary after a read operation.

The design of DRAM is optimized for density. Its cost per bit is much lower than SRAM's which allows it to reach a much higher capacity under the same chip area. Although it does not require continuous power supply to maintain its content, periodical refresh operation is necessary as the charge stored in the capacitor will gradually fade out. DRAM refresh not only consumes extra energy,

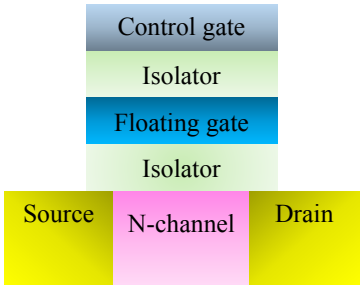


Fig. 2.2 A N-channel floating gate transistor (FGMOS).

but also affect performance since the normal operation must be blocked when the cell being refreshed.

2.1.3 Read-only Memory and Flash Memory

Read-only memory (ROM) is a class of non-volatile memory (NVM) that can store data permanently without power supply. Under most of the circumstances, the only operation to ROM is read. Thus ROM is optimized for read operation, and typically requires a very different procedure of reprogramming its contents. Such a procedure can be either physical as in *erasable programmable read-only memory* (EPROM) or electrical as in *electrically-erasable programmable read-only memory* (EEPROM).

Flash memory originated from EEPROM. It is based on *floating-gate* MOSFET (FGMOS) which resembles as a capacitor. In FGMOS, there is a secondary gate (FG) deposited under the control gate (Figure 2.2). FG is completely electrically-isolated, hence the charge (data) trapped inside can be kept unchanged without power supply. An electric field is applied to remove the charges and place electrons into the FG.

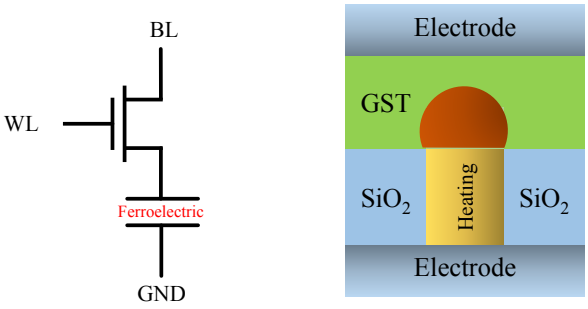
There are two types of flash memory: *NOR*-type and *NAND*-type. They differ in the way of connecting the cells in forming a data array. NAND-type flash is denser than NOR-type and cost less, so it dominated most of the large-capacity storage systems. Note that NAND flash is not a random-addressable memory. Its operating mechanism is similar to a block device where reading and programming is performed on a page basis while erasing on a block basis. In addition, reprogramming of a dirty block requires erasing of the existing contents first. Therefore, flash memory operates at a much slower speed than other volatile memories such as SRAM and DRAM, and it is typically deployed at the bottom of the memory hierarchy where the access frequency is relatively low.

2.2 Emerging Memories

Due to their desired features, a number of emerging memory technologies are currently being investigated as promising candidates on mitigating the issues posted by conventional memories, especially SRAM and DRAM.

2.2.1 Ferroelectric RAM

Ferroelectric RAM (FeRAM) is a random-access memory that has similar structure as DRAM but uses ferroelectric layer instead of dielectric layer to achieve non-volatility [46]. It adopts a ferroelectric capacitor in a 1T-1C structure as shown in Figure 2.3a. The ferroelectric material has the effect of semi-permanent electric dipoles formed in the crystal structure. The content stored is recognized by the polarization state of the dipoles. Programming of the cell is done through applying an external electric field across a dielectric. Note that although the



(a) 1T-1C ferroelectric cell (b) Phase change cell

Fig. 2.3 The cell structures of FeRAM and PCM.

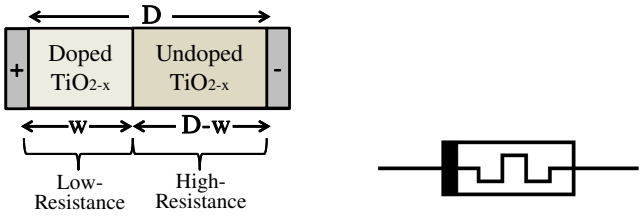
dipoles can retain their polarization state after programming, the read operation is still a destructive process and re-written of the data is necessary.

2.2.2 Phase-change Memory

Phase-change memory (PCM) is a type of NVM that exploits the unique behavior of chalcogenide glass [7]. Shown in Figure 2.3b, the cell is built with active material including *Ge*, *Sb* and *Te* (GST) that are sandwiched between two electrodes. GST exhibits two meta-stable phases (states) with different resistances: a poly-crystalline and an amorphous phase. The phase change (programming) is done through a thermally-induced transition (temperate change) by heating up the bottom resistive electrode.

2.2.3 Memristor

Memristor is the fourth fundamental passive circuit element which is firstly demonstrated by Hewlett-Packard in 2008 [80]. It was predicted to exist in nature during last 70s [21], but was only found 30 years later. Memristor is a bipolar



(a) Internal structure (b) Electronic symbol

Fig. 2.4 The TiO_2-based memristor.

device that consists of a semiconductor thin film of thickness D sandwiched between two metal contacts (Figure 2.4). A movable boundary (domain wall) separates the mental oxide layer into two regions: a highly conductive doped region (width w), and a high resistive undoped region (width $D-w$). In HP's implementation, the undoped region contains oxygen-rich TiO_2 while the doped region is injected with auxiliary oxygen vacancies, and is thus positively charged. By applying a positive voltage to the electrode on the doped (undoped) side, the oxygen vacancies move towards the undoped (doped) region. As a consequence, the boundary front shifts, and the overall resistance of the device is reduced (increased). When the metal oxide layer becomes completely doped (undoped), the device enters a hysteresis state with lowest (highest) resistance. The process is reversible. By applying negative voltage to the doped side, the oxygen defects can be pushed back to the doped region and the overall device resistance would increase accordingly.

Formally, given the width w of the doped region, and a device thickness of D, the resistance of a current-controlled memristor can be expressed by

$$R(w) = R_{on}\frac{w(t)}{D} + R_{off}(1 - \frac{w(t)}{D}) \tag{2.1}$$

where R_{on} and R_{off} are the minimum and maximum resistance as the metal oxide region becomes completely doped and undoped, respectively. Moreover, the changing rate of w is represented by

$$\frac{dw(t)}{dt} = \mu_v \frac{R_{on}}{D} i(t) \tag{2.2}$$

where μ_v is the average ion mobility, and $i(t)$ is the current flow across the device at time t.

Memristors can be used as single-level cell memory through mapping their resistance states into binary bits, say with the high-resistance state corresponds to logical 0 while low-resistance state corresponds to logical 1. More interestingly, the device's resistance can be programed into any intermediate value by a careful-controlled voltage and timing. In another words, it has the potential to function both as single- or multi-level cells.

Other than HP's TiO_2-based implementation, several types of materials have been shown to exhibit a similar characteristics [43, 57], and hence can be used in the fabrication of memristor. In our study, we choose HP's implementation model since it has received more attentions than the others with several practical simulation models widely available [11, 68, 49, 58].

2.2.4 Spin-transfer-torque RAM

Spin-transfer torque RAM (STT-RAM) belongs to the family of *magneto-resistive* RAM (MRAM) which stores data as magnetic resistances. The name of STT-RAM comes from a physical technique called *spin transfer switching* [25] and it has been proved to be more superior than other types of MRAM. The basic

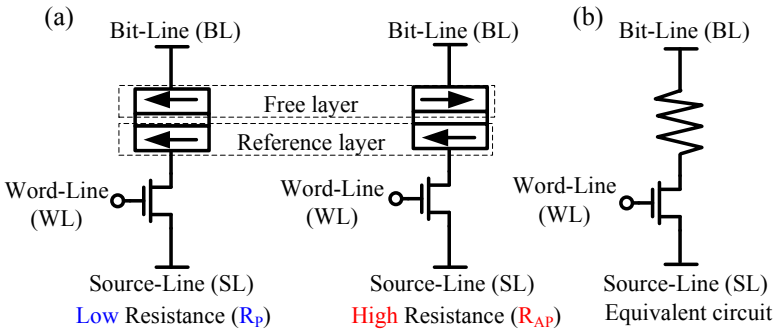


Fig. 2.5 Typical structure of a 1T-1MTJ STT-RAM cell.

data storage unit inside a STT-RAM cell is the *magnetic tunnel junction* (MTJ). Figure 2.5 shows a typical one transistor, one MTJ (1T-1MTJ) STT-RAM cell structure. The MTJ has two ferromagnetic layers that are separated by an oxide tunnel barrier. The ferromagnetic layer at the bottom (the *reference* layer) comes with a fixed magnetization direction while that for the top layer (the *free* layer) can be changed though a spin-polarized current. This configuration is known as a *spin valve*.

When the magnetization direction of the free layer is in *parallel* (P) or *anti-parallel* (AP) to the reference layer, the device is in low- or high-resistance state, representing logical '1' or '0'. As such, the data is encoded by the resistance of the MTJ. For read operation, the cell is first selected through the *word-line* (WL) to the gate of the nMOS transistor, a small sensing current is injected into the MTJ from the *source-line* (SL). The resistance state of the device is then determined by comparing the voltage output in *bit-line* (BL) to a reference value. To program (write) the device, a current larger than *critical switching current* (I_C) is applied from BL to SL. The polarization of the injected current determines how the magnetization direction of the free layer would be switched (P or AP to the reference layer).

Volatility-relaxed STT-RAM

When compared to volatile SRAM, a typical storage-class STT-RAM which can sustain data at least 10 years at room temperature incurs a up to 20× programming latency [98]. Theoretically, the data retention time T_{ret} of a MTJ can be formulated as

$$T_{ret} = \frac{1}{f_0} e^{\Delta} \tag{2.3}$$

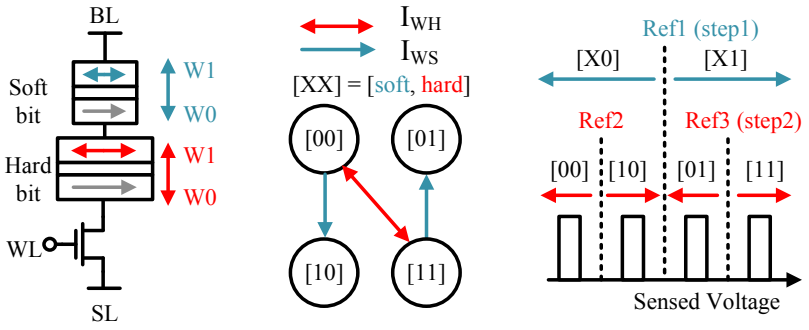
where Δ is the magnetization stability height and f_0 is the thermal attempt frequency. For storage purpose, f_0 is taken from the order of GHz. The value of Δ is determined by

$$\Delta = \frac{K_u V}{k_B T} \tag{2.4}$$

where K_u is MTJ characteristic constant and k_B is Boltzmann constant [25]. T is the device working temperature and V is the effective activation volume. Hence, the retention time of a MTJ decreases exponentially to the cell surface area, and as the working temperature rises. Previous study proposed that by reducing V, the MTJ switching current density can be decreased so as the write latency and energy required [75]. As a consequence, retention time T_{ret} is inevitably be reduced and a refresh scheme is necessary to maintain data integrity. Such kind of volatility-reduced STT-RAM trades the write performance/energy for extra refresh penalties.

Multi-level cell STT-RAM

To further improve the storage density, *multi-level cell* (MLC) STT-RAM has been investigated. It is implemented by placing more than one MTJs in a cell. One



(a) Cell structure. (b) Write operation. (c) Read operation.

Fig. 2.6 MLC STT-RAM cell structure and operations.

of the conventional MLC STT-RAM structure stacks two MTJs vertically [36], and is more robust against process variation [98]. Figure 2.6 demonstrates the cell structure and read/write procedures.

The two MTJ must be of different sizes in order to differentiate two logic bits in one cell. We name the data stored in the small and large MTJs as *soft-bit* and *hard-bit*, respectively, as shown in Figure 2.6a. In a given magnetic process, both the *resistance-area product* (RA) and the *critical switching current density* (J_C) remain constant, therefore the soft-bit has a larger resistance but requires a smaller switching current I_C than the hard-bit.

Programming an MLC requires a two-stage operation as shown in Figure 2.6b. First, a strong current $I_{WH} > I_{C,Hard}$ is applied which switches both the hard-bit and the soft-bit. Then in the second stage, a smaller current satisfying $I_{C,Hard} > I_{WS} > I_{C,Soft}$ is used to switch only the soft-bit. Similarly, reading out the data from a MLC STT-RAM requires two steps. A reference voltage ref–S is first used to detect the soft-bit. According to the result, a second reference voltage ref–H or ref–H′ is used to sense the hard-bit, as shown in Figure 2.6c. Therefore, reading and writing the MLC takes more time than a SLC. Note that

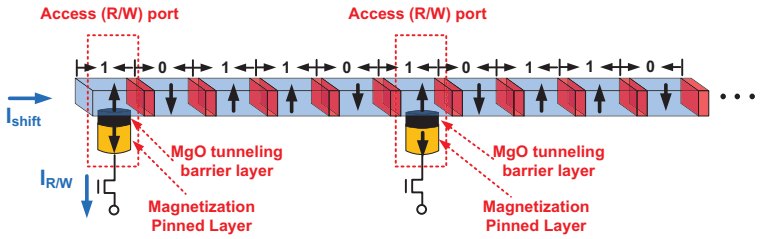


Fig. 2.7 The horizontal structure of a domain-wall memory [83].

if the hard-bit is fixed at a value, reading and writing the soft-bit requires only a single step with smaller current densities. Thus, MLC STT-RAM can operate as SLC to accelerate the access speed by reducing half of the capacity.

2.2.5 Domain-wall Memory

Domain-wall memory (DWM), or *racetrack memory*, stores multiple data bits in the domains on a single ferromagnetic nanowire [83]. Figure 2.7 shows a DWM macro-cell that has a similar structure to a tape device. These domains are separated by ultra-thin domain walls and each of them has its own magnetic direction that can be individually accessed/programed. Under such an organization, a read/write operation requires three step. First, the particular domain to be accessed is shifted to the access port. Then the required operation (R/W) is performed by applying an appropriate current to the MTJ built at the access port. Upon the desired operation is completed, the domain must be shifted back to its original location in order to maintain a proper addressing. Such a memory can be easily made in very high capacity by arranging multiple racetracks (tapes) together either horizontally or vertically. The main drawback is that the access time depends on the relative location of the data bit and the access port.

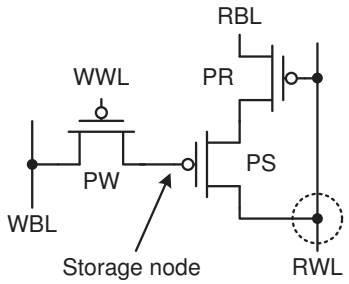


Fig. 2.8 A logic-compatible gain cell eDRAM.

2.2.6 Embedded DRAM

Embedded DRAM (eDRAM) shares similar structure with DRAM but can be integrated into the same die with processor cores as on-chip memory. There are two types of eDRAM: 1T-1C cell (Figure 2.1b) and logic-compatible gain cell (Figure 2.8). 1T-1C cells are denser but the read operation is destructive while gain cell has a better read performance and can be built using standard CMOS technology. However, both types still require periodical refresh which incurs additional energy and performance overheads.

3D XPoint Memory

Recently, Intel has proposed a new type of memory technology called *3D XPoint* [1]. It is a resistance-based memory that relies on different resistance levels to differentiate logical '0' and '1'. While not much details of the technology has been disclosed, Intel claimed it is more reliable than NAND flash with a comparable access speed to DRAM. Further investigation showed that this technology is using PCM and targeting on big data applications. Thus it shall replace neither NAND flash nor DRAM.

	SRAM	DRAM	FeRAM	PCM	Memristor	STT-RAM	DWM
Density	Low	High	Medium	High	High	Medium	High
R/W speed	Fast	Fast	Medium / Slow	Fast / Slow	Medium	Fast / Medium	Fast / Medium
R/W energy	Low/ Low	Low/ Low	Low / High	Low / High	Low / Medium	Low / Medium	Low / Medium
Volatile	Yes	Yes	No	No	No	No	No
Endurance	High	High	Medium	Medium	Medium	High	High

Table 2.1 Comparison of memory technologies.

2.3 Summary

Figure 2.1 summarizes the characteristics of several memory technologies. Each of them has its own advantages and disadvantages. For instance, SRAM has the highest performance but is low density; DRAM (eDRAM) can be easily made in large capacity but requires periodical refresh for maintaining the data; PCM has similar density as DRAM but the endurance is few orders of magnitude lower.

Figure 2.9 shows the memory hierarchy and the suitable technologies according to the existing states. At higher levels in the memory hierarchy, components such as buffers and caches are more performance critical. They exhibit strong *locality of reference* so that capacity is less important here. As we goes down in the memory hierarchy, performance becomes less important and cost-per-bit starts to play the main role. Due to such a property, on the conventional side, SRAM is usually chosen to implement some higher level memory components while DRAM dominates the area of main memory.

On the emerging memories front, STT-RAM, DWM and eDRAM has been widely proposed as replacements of SRAM-based on-chip caches [82, 75, 45, 37, 83, 14] while PCM is often treated as a replacement of DRAM in constructing

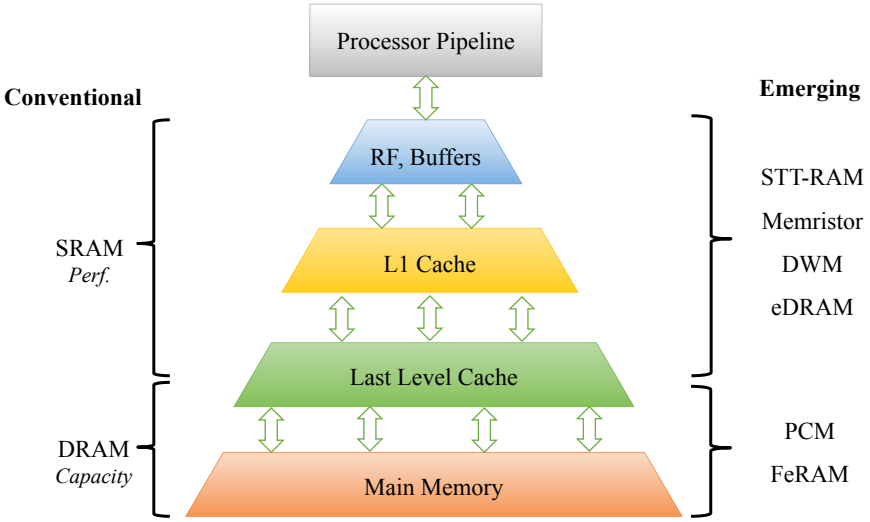


Fig. 2.9 Memory hierarchy and memory technologies.

off-chip memories [60, 23]. More interestingly, memristive devices has been used in the realization of large-scale neuromorphic system [43, 76]. These previous studies indicate that emerging memories shall play an important role in the design of future energy-efficient memory hierarchies.

Chapter 3

Architecting Memristor as Low-Power Analog Neural Branch Predictor

In this chapter, we examines the use of emerging memories at the highest level in the memory hierarchy. Example memory components at this level includes *register files* (RF), *translation lookaside buffer* (TLB), *return-address stack* (RAS) and *dynamic branch predictor*. At this level, most of the components are optimized for latency since the operations on them are very likely to be on the critical path. For example, register read, address translation and branch prediction. Figure 3.1 shows the scope of this chapter and the memory technologies involved.

This chapter will demonstrate a low-power neural branch predictor design constructed with memristors. By the help of analog computing techniques, as well as exploiting the accuracy tolerance inside branch prediction, our predictor

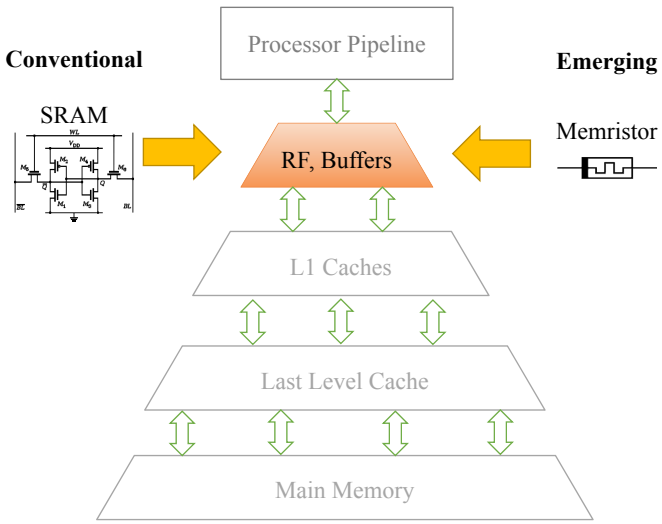


Fig. 3.1 Scope and memory technologies used in this chapter.

is able to efficiently realize a neural prediction algorithm with a better accuracy and a lower energy consumption than digital-based predictors.

3.1 Branch Prediction

Modern processors are usually deep-pipelined with the capability of speculative execution. Thus it makes branch prediction a crucial factor in single-thread performance. For instance, a x86 uop branch mis-prediction incurs at least 15 clock cycle penalty measured in the pipeline of Sandy Bridge microarchitecture [32]. Compared to a typical arithmetic operations that usually takes only a few cycles to complete, each branch mis-prediction creates a large bubble in the instruction flow and reduce effective performance. Figure 3.2 shows how the pipeline operates when: a) no branch predictor; b) branch mis-predicted; c) branch correctly predicted. Here we assume each branch instruction takes 10 clock cycles to

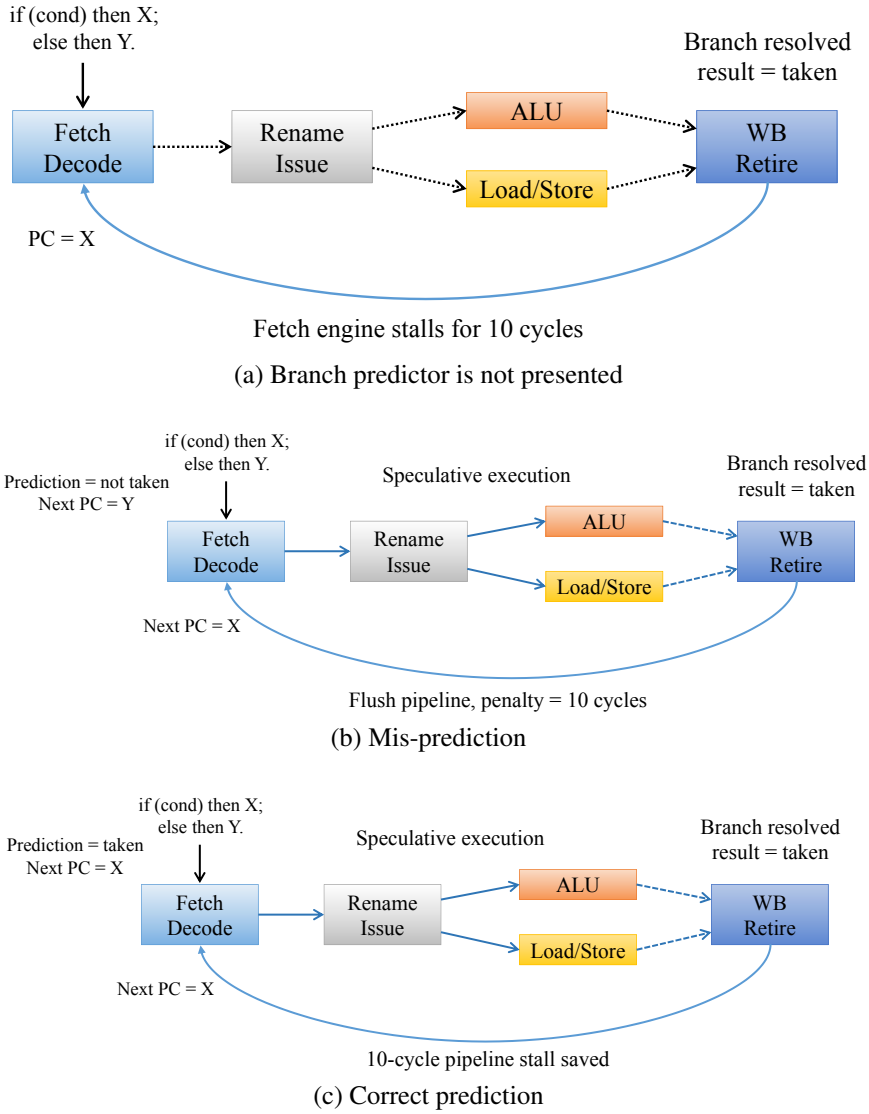


Fig. 3.2 The importance of branch predictor.

resolve. In Figure 3.2a, no prediction mechanism is presented so the pipeline has to stall for 10 cycles until the branch gets resolved. In Figure 3.2b, the branch is mis-predicted. It takes roughly same amounts of time to flush the incorrect instructions from the pipeline. In Figure 3.2c, the branch is correctly predicted. Hence, the speculative execution path can be finalized, saving the 10-cycle stall.

Types of branch predictor

In general, branch predictors can be categorized into either *static* or *dynamic*. Static predictor makes predictions based entirely on static information such as branch target and type of the branch. Dynamic predictor takes additional runtime information into account including branch pattern history so thus resulted in much better prediction accuracy. In the past twenty years, numerous dynamic branch prediction schemes have been introduced. Among them, *two-level adaptive branch predictor* [96, 97, 55] is the most widely used approach offering high prediction accuracy with moderate implementation cost. However, the resource requirement for two-level predictor grows exponentially when the history length increases. Hence, the effectiveness of all two-level-based schemes are limited by how many branch history patterns they can recognize.

Neural branch prediction

It's worth noting that branch prediction is essentially a machine learning problem. The predictor gathers information from the retired branches and learns the branch behaviors (patterns) based on some statistics models. Then it makes predictions (inferences) for the future branches using its existing knowledge and trains the model when necessary. Therefore, some researchers have proposed neural methods in the design of branch predictors [78]. In particular, the *perceptron predictor* claimed to be the first neural branch predictor that is deemed feasible for actual deployment [41] and it forms the base of subsequent neural-inspired predictors [38–40]. Nonetheless, it incurs a much longer prediction latency when

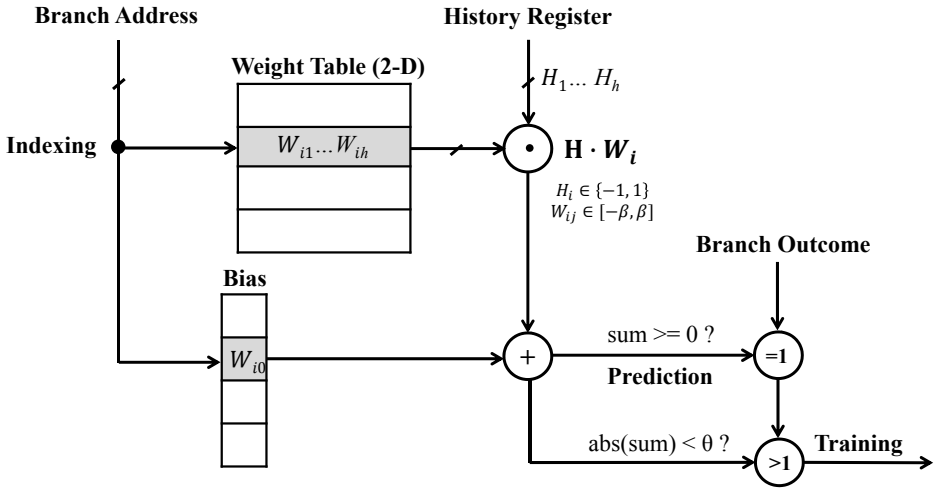


Fig. 3.3 The perceptron predictor diagram.

compared to other two-level-based predictors. The following section describes the implementation of a perceptron predictor in details.

3.1.1 Perceptron Predictor

The design diagram of a perception predictor is shown in Figure 3.3. Its algorithm requires a two-dimension weight table with an extra column of bias weight to be stored in some fast SRAMs. Each row of the table consists of h n-bit signed integer weights, where h is the global history length to be considered.

When a branch is encountered, its address is used to index into one of the rows, and a dot-product is performed between the global branch history vector (with '1' being taken to mean 'taken', and '-1' for 'not-taken') and the selected row (excluding the bias weight). The prediction is generated by examining the sign bit of the sum of the dot-product with the bias weight. Mathematically, this

weighted sum S on a selected row i is calculated by

$$S = W_{i0} + \sum_{k=1}^{h} (W_{ik} \times H_k) \tag{3.1}$$

where W_{i0} denotes the bias weight of row i, $(W_{i1}, \ldots, W_{ih})$ is the selected weight vector and $H = (H_1, \ldots, H_h)$ is the h-bit history register.

Training of the weight table occurs when the prediction is wrong or the absolute value of S is less than a pre-defined threshold θ (meaning that the prediction is weak). The detailed algorithm is described in Algorithm 1. Note that all arithmetics here are saturated. The rationality behind the training is that when the outcome of the current branch is strongly correlated with a historical branch, the magnitude of the weight should be large enough to have a notifiable impact on the weighted sum. Otherwise, the weight will be close to zero which contribute very less to the calculation of S. Since it requires only one column in the weight table for each historical branch, perceptron predictor can consider much longer history than other types of predictor such as two-level predictor within the same storage budget.

3.1.2 From Digital to Analog

Despite of the high prediction accuracy and a more efficient use of memory storage, perceptron predictor (and other neural-based schemes) comes with a serious drawback: the time to perform a dot-product limits how fast a prediction can be made. By using an adder tree, adding h integer numbers still requires $\log h$ operations, not to mention the complexity of the extra circuitry supporting parallel addition. Although there are proposals to speed up the computation [38],

```
if prediction != outcome OR S < θ then
    if outcome == taken then
        W_i0 ← W_i0 + 1
    else
        W_i0 ← W_i0 − 1
    end
end
forall the j in parallel do
    if H_j == outcome then
        W_ij ← W_ij + 1
    else
        W_ij ← W_ij − 1
    end
end
```

Algorithm 1: Training the perceptron predictor for branch i

they come at the expense of accuracy. Due to these issues, most neural-based schemes to date have been deemed impractical for actual implementation.

Analog Current Summation. While adding a long list of weights is slow and expensive in the digital domain, there is a fast and efficient analog circuit technique which can achieve similar effect. *Kirchoff's Current Law* (KCL) states that at any junction in an analog circuit, the sum of currents flow in equals to the sum of currents flowing out. Mathematically, given a junction that have i currents passing through, we have

$$\sum_{i=1}^{k} I_i = 0 \tag{3.2}$$

Note that current I is a signed quantity where a positive value represents the current that flows into the junction while a negative value represents the current that flows out.

By the property of KCL, connecting a list of analog currents with the same direction would produce a current sum with an magnitude that equals to the sum of all the currents. In theory, such an operation is equivalent to the addition of weights which are now represented by analog currents.

Analog Neural Predictor. Since neural-based computations in nature are essentially analogue, researchers have introduced analog methods into neural predictors [77]. The main purpose here is to shorten the critical path by replacing the high-latency and energy-consuming digital addition with a fast and efficient analog current summation. While storing analog signal is difficult, the present of digital-analog-converter (DAC) is necessary to convert between digital weights and analog currents. Hence, the requirement of SRAM storage is not eliminated.

Integrating Storage and Computation. With the discovery of memristor, it has prompted us that the functions of memory and computation can be combined in a single device within a neural system [43]. Hence, we took the benefits from the memristor and have worked out a new design of neural branch predictor.

3.2 Memristor-based Branch Predictor

This section describes our memristor-based neural predictor design. We shall present the high-level architecture followed by the internal circuit design of the memristor cell.

3.2.1 Architecture Design

Figure 3.4 shows the architecture-level design of our memristor predictor. Our design is based on the perceptron predictor described in previous section. How-

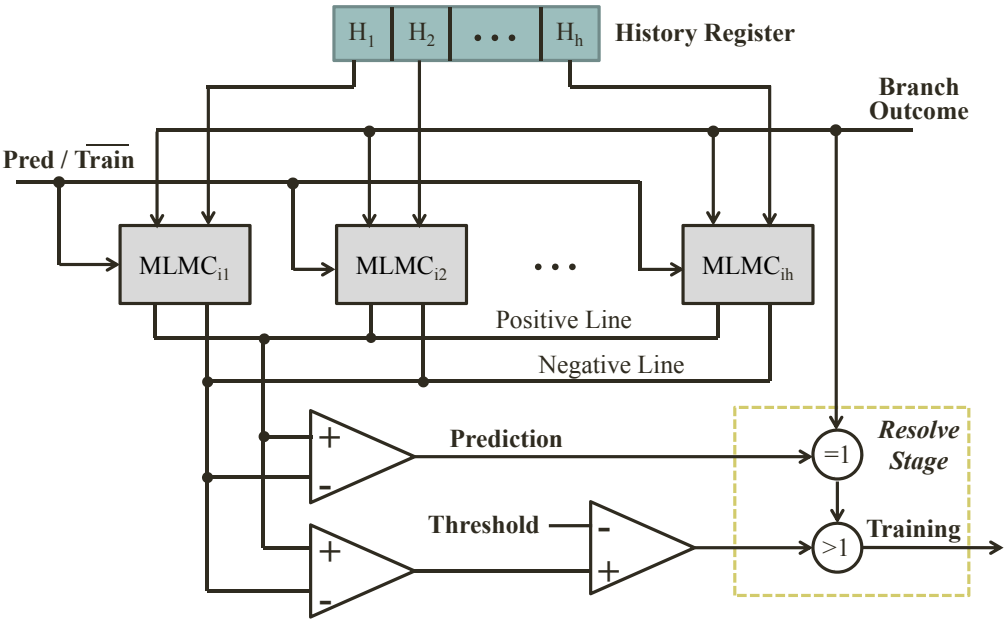


Fig. 3.4 Architecture design diagram of the memristor-based neural predictor

ever, instead of using conventional SRAM storage, each weight is now stored in a two-dimensional table of *multi-level memristor cell* (MLMC). The MLMC at position (i, j) has two analog outputs: P_{ij} and N_{ij}. The weight W_{ij} stored in this MLMC is effectively the relative difference between them, i.e., $W_{ij} = P_{ij} - N_{ij}$. By simply exchanging the roles of P_{ij} and N_{ij} inside a MLMC the weight can be negated. This is controlled by the history bit, H_j. Such a design allows us to perform multiplication by 1 and -1 in a faster manner than the digital counterpart in which a complement operation is needed.

Similar to the perceptron predictor, the program counter (branch address) is used for indexing a row of weights: $W_{i1}, \ldots, W_{ih}$ during the prediction phase. Each MLMC will output one of its analog current signal on a positive line and a negative line in accordance to the *global history register* $(H_1, \ldots, H_h)$ in the

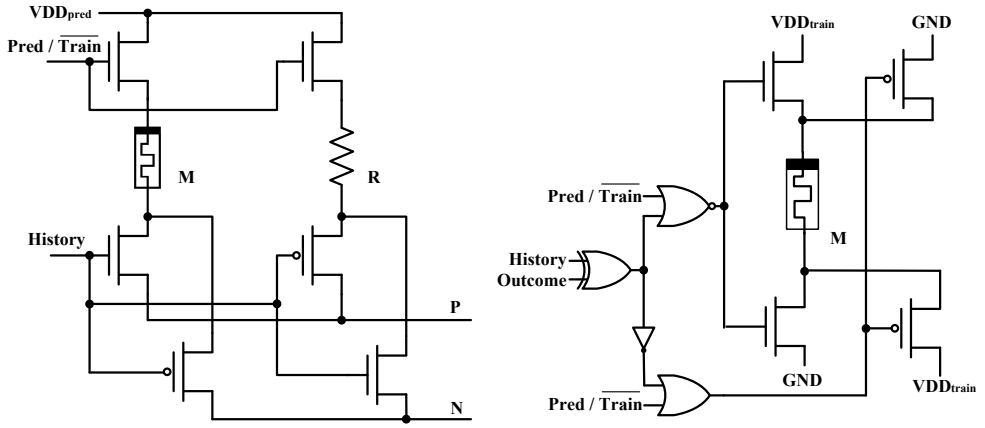
following way. If $H_k = 1$, P_{ik} will be put onto the positive line and N_{ik} will be output onto the negative line. If $H_k = 0$, the roles of N_{ik} and P_{ik} are reversed.

All the current signals on the shared positive (negative) line will be summed up naturally by Kirchoff's Current Law. The final prediction is made based on the relative current difference between the two lines, which is detected by an *latched comparator* [30]. If the total current on the positive line is equal or larger than that on the negative line, the branch is predicted to be 'taken'. Otherwise, it is predicted to be 'not-taken'. Note that the analog summation process is accomplished almost instantly once all the cell outputs stabilize. This provides a significant speed advantage over the digital adder tree method.

The training signal is generated at the branch resolving stage of the pipeline. To avoid a second read of the perceptron table, we shall perform a threshold comparison at the prediction stage and propagate its result to the later stage where branches are resolved. The comparison is done by first computing the difference between the analog signal on the positive and negative lines, followed by a latched comparison. This is done in parallel with the prediction. While training is required, all MLMCs in the selected row would be updated simultaneously given the branch result and history bits.

3.2.2 Circuit Design

Figure 3.5 gives the detailed design of a MLMC. We can divide the whole cell into two sub-branches: a memristor branch, and a resistor branch. The value of R is chosen to be in the middle of the resistance range of M. When the prediction signal is activated, the upper CMOS gates saturate, and the currents



(a) Prediction circuit (b) Training circuit

Fig. 3.5 Circuit design inside a MLMC

going through M and R would be directed to output terminals P and N based on the history bit. The history input controls the two pairs of MOSFET below M and R. When the history is '1' (taken), M is connected to P and R is connected to N. If the history is '0' (not-taken), the roles of P and N are reversed which is equivalent to multiplying the weight by -1.

For the training part, auxiliary CMOS gates are added to control the programming current/voltage's direction into M1. When the branch history and outcome are positively correlated, the middle pair of MOSFETs are activated so that the current will push the domain wall towards the undoped side, and so decreases the resistance of M1. On the other hand, if the branch history and outcome are different (negatively correlated), the right pair of MOSFETs are activated to enable a reverse current for pushing the domain wall to the doped region, thereby increasing the device resistance.

Notice that the magnitude of the prediction (read) and training (write) voltages are different. To perform predictions, a small V_{DD} is chosen so that it brings only

a negligible disturbance to the memristor state while a larger V_{DD} is required for proper programming of the cell.

In theory, the memristor can be configured with arbitrary resistance levels to imitate a n-bit number. However, using more levels would result in lower noise margins, which may make the output difference undetectable. For practical reasons, we configure the memristor to have only 16 levels of resistances, equivalent to a digital design using 4-bit weights.

3.2.3 Discussion

Analog noise and thermal fluctuation

Unlike digital circuits, analog systems, especially for those on-chip components, are more susceptible to signal noises. The current signals on the two shared lines can be disturbed by voltage bounces generated from MOSFET switchings which may affect the results of the analog computations. However, this can be mitigated through some mixed-signal IC design techniques such as the *guard rings* [81]. On the other hand, instead of using conventional SRAM storage as in SNAP [77], our design stores the perceptron weights in a purely analog form (resistance). The elimination of current-steering DACs potentially reduces the amount of noises coming from signal conversion, and thus enhances the overall robustness.

While the ion mobility inside a memristive material like TiO_2 may be affected by thermal fluctuations, there is no evidence currently to show that this issue is significant enough to have major impact on our design. As more and more materials are found to exhibit memristive property, it is likely that a commercialized

memristor device will be as temperature-insensitive as the other on-chip CMOS components.

Memristor latency and state-drift

Since the memristor is essentially a variable-resistor, the read latency of a MLMC is dominated by the switching speed of the MOSFETs, and hence has very limited impact on the prediction latency. On the other hand, the original TiO_2-based memristor [94] has an unfeasibly long programming latency under a typical V_{DD}. Such a limitation should eventually be addressed by technological advances such as the use of recent discovered TaO-based materials [57], which has already demonstrated sub-nanosecond switching speed.

Another issue related to any memristor-based design is the *state-drift* problem. Since the existing ion-drift models do not impose a programming threshold, even a tiny electric charge may disturb the memristor's state and such an effect can be accumulated. Fortunately, recent research has shown that the internal switching dynamics requires a fairly strong electric field for programming in practice [66]. This means that as long as the sensing current is kept under a threshold, the state-drift issue shall not occur.

Process variation

As the fabricating process continues to scale, process variation becomes critical, and can affect some key design parameters of nanoscale devices. For the memristor, fluctuations in the resistance range directly impacts the accuracy of the analog computation. Although a neural-based predictor is more tolerant to minor weight variations, these manufacturing fluctuations could conceivably lead

to performance degradation in some cases. Rajendran et al. [69] have analyzed the effect of process variation on a memristor-based threshold gate design, and proposed two algorithms to correct these variations. Their solutions can be essentially applied to our design and are out of the discussion of this work.

3.3 Evaluation

This section evaluates the effectiveness of our predictor design and compare it with the existing neural and non-neural schemes.

3.3.1 Experiment Setup

To evaluate our design, 20 benchmarks from the SPEC2006 suite are simulated using the cycle-accurate simulator MARSSx86 [64]. Cycle-accurate simulation evaluates not only the prediction accuracy, but also the performance impact under a modern processor setting. The detailed parameters for the base machine are shown in Table 3.1. The accuracy of the branch predictor is measured in *mis-predictions per kilo-instruction* (MPKI), which is a common metric for evaluating branch predictors. As for comparison, we have chosen two other neural predictors and one non-neural predictor in our experiments:

- **GShare**. This is a variant of the original two-level predictor that can be found in many commercialized processors [5]. It features a relatively high prediction accuracy and a low implementation complexity. As our baseline predictor, we have chosen a storage budget of (128K + 16)bits configured as 64K $\times$ 2-bit pattern history table with 16-bit global history.

- **Perceptron**. This is the neural predictor described in section 3.1.1. We choose 4-bit digital weight width in order to provide a fair comparison with the memristor-based predictor. Under the same storage budget as gshare, various global history lengths are tested and the best-performed one is chosen (48-bit).

- **SNAP**. This is a neural-inspired predictor that features an analog implementation [77]. The scaling factors are calculated using the original fitting formula. While the original scheme features several weight tables of different dimensions, we simplified the design with a single weight table. The global history register is fixed at 48-bit as well.

- **Memristor**. This is our proposed design. The memristor model used is a simple linear ion drift model [11] that has been configured with 16 distinct resistance levels. The double-memristor predictor scheme proposed in [34] was also evaluated. The weigh table dimension and history length are chosen to be the same as perceptron and SNAP, although memristor-based storage is considered in higher density.

Despite that the accuracy of a linear ion drift model is usually considered to be insufficient in modeling fabricated memristor device, it satisfies the basic memristive system equation and has the advantage of being computational efficient [11]. On the other hand, while the TaO_2-based memristor has already demonstrated sub-nanosecond switching performance, the lack of a practical model makes the simulation difficult [57]. Therefore, some parameters in the linear ion drift model are tuned manually so that the device can work according to a GHz-level clock frequency, assuming these physical properties of memristor are

Parameters	Value
Processor Core	Single, out-of-order
Pipeline Width	4
Pipeline/Frontend Stages	14 / 4
Fetch/Issue Queue Size	48 / 64
ROB/Register File Size	128 / 256
Load/Store Queue Size	48 / 32
I-TLB/D-TLB Size	64 / 64
In-flight Branches	24
BTB Size	4K, 4-way associative
RAS Size	24
Functional Units	4 ALU, 2 FPU, 2 LU, 1 SU
Memory/Cache System	Perfect cache, 3-cycle latency

Table 3.1 Simulation platform.

scalable. Although modern fetch units may withstand the penalty of multi-cycle prediction in certain situations, for the simplicity of discussion, we assume all predictors generate a prediction within one CPU cycle.

3.3.2 Energy Consumption

We measured the energy consumption by calculating both the static energy (leakage) and the dynamic energy consumed while performing branch predictions and predictor updates (training).

Leakage energy

The leakage energy consumed by storing the weight table was estimated through the modeling of a tagless table. CACTI simulation shows that a 128kbits SRAM table consumes 7.9mW of leakage power (data cells only). Such an amount of leakage power shall be similar for all kinds of SRAM-based predictors.

On the analog side, we used LTSpice with 45nm PTM to monitor the current passes through each MLMC for measurement of the power consumption. Simulation showed that each MLMC (single-memristor) drains 5.5nW of standby power, and thus the same-sized weight table consumes 67.6μW of leakage power in total. For comparison, the double-memristor design [34] consumes a little bit more than 92μW due to the extra MOSFETs required for programming two memristors. Hence, our approach achieves a two orders of magnitude savings in leakage power compared to those SRAM-based predictors. Note that due to the significance of leakage power, the static energy has already dominated overall energy consumption in the case of perceptron and SNAP.

Dynamic energy

The dynamic energy consumed by a prediction includes the table lookups and the analog/digital computation. Given the same table dimensions, the lookup step consumes a similar amount of energy for both analog and digital storage. For a SRAM-based table, the energy required to read out a 24-byte block (48 history counters of 4-bit weight) was measured at 18.9pJ. At a 1 GHz clock with a 1V V_{DD}, our analog predictor requires only 0.3pJ to obtain stable outputs on both the positive and negative lines. The alternative double-memristor scheme consumed a slightly lower 0.27pJ of read energy. For comparison, SNAP required a larger 0.4pJ energy just for its analog computation.

For training (i.e., updates), our single-memristor design consumes around 0.82pJ while the double-memristor scheme consumes more than 1.87pJ. Note that the SRAM-based perceptron/SNAP consumes similar energy for both reads and updates. Figure 3.6 shows the total average energy breakdown for the two

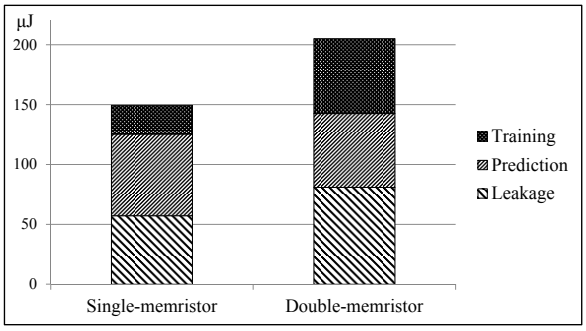


Fig. 3.6 Energy breakdown for the two memristor predictors.

memristor-based schemes, averaged from all the benchmarks tested. Our single-memristor design outperforms the double-memristor approach in both leakage and dynamic energy (prediction + training). The data also shows that the double-memristor design results in a slightly better prediction energy, but requires more energy for training. In total, our design reduces energy consumption by 27.3%.

3.3.3 Prediction Accuracy

Figure 3.7 shows the prediction accuracy of gshare, perceptron, SNAP, and our memristor-based predictor. In certain cases like `410.bwaves`, `444.namd` and `470.lbm`, more independent looping-behavior branches are present. Thus gshare performs better than perceptron in separating those interferences from unrelated histories. However, on average, gshare is not as accurate as the neural-based predictors. The perceptron predictor generally improves the low accuracy of gshare. In particular, for integer benchmarks like `401.bzip2`, `456.hmmer` and `471.omnetpp` that usually contain lots of complex dependent branches, larger reductions in MPKI were recorded because the perceptron algorithm can capture longer historical branch relationship than gshare.

When compared to digital perceptron, the analog counterparts turns out to be slightly more accurate. SNAP improves on the perceptron's weakness in predicting simple uncorrelated looping behavior branches by reducing the interference from unrelated historical branches. Our memristor-based scheme, on the other hand, behaves more nonlinearly than either digital perceptron or SNAP, due to memristor's characteristic and its analog computation. This allows it to resolve certain irregular dependent branches. For example, compared to gshare, our predictor reduces by 35% the MPKI for `473.astar` which is the most difficult benchmark to predict. Another case with many irregular branches is `403.gcc`, where the memristor predictor reduces the MPKI by 24%.

3.3.4 Performance Analysis

The normalized IPC rates against the baseline are shown in Figure 3.8. For most benchmarks, a lower MPKI usually increases the IPC rate. However, all three predictors suffered larger than 13% IPC drops compared to gshare in the case of `429.mcf`. The digital perceptron only increased MPKI by a marginal amount, and the other two had lower MPKIs. Similar results were observed in `470.lbm` where the IPC for SNAP and memristor predictor decrease by more than 11% even with better MPKI numbers. Such a phenomenon shows that not all mis-predicted branches incur the same performance penalty. Different predictors may predict the same branch differently, even if they produce similar MPKI numbers. In certain cases, while neural-based predictors predict more branches correctly, they also mis-predicted more critical branches than gshare. Fortunately, the impact on performance from those critical branches is not significant. On average, digital

perceptron, SNAP, and our memristor predictor boost the baseline IPC rate by 7.1%, 8.1% and 8.6%, respectively.

Compared with the double-memristor proposal [34], our predictor reduces the MPKI by more than 10% with a 3% increase in IPC, under the same weight table dimension. Therefore, combined with the improvement in energy consumption, our single-memristor approach is superior.

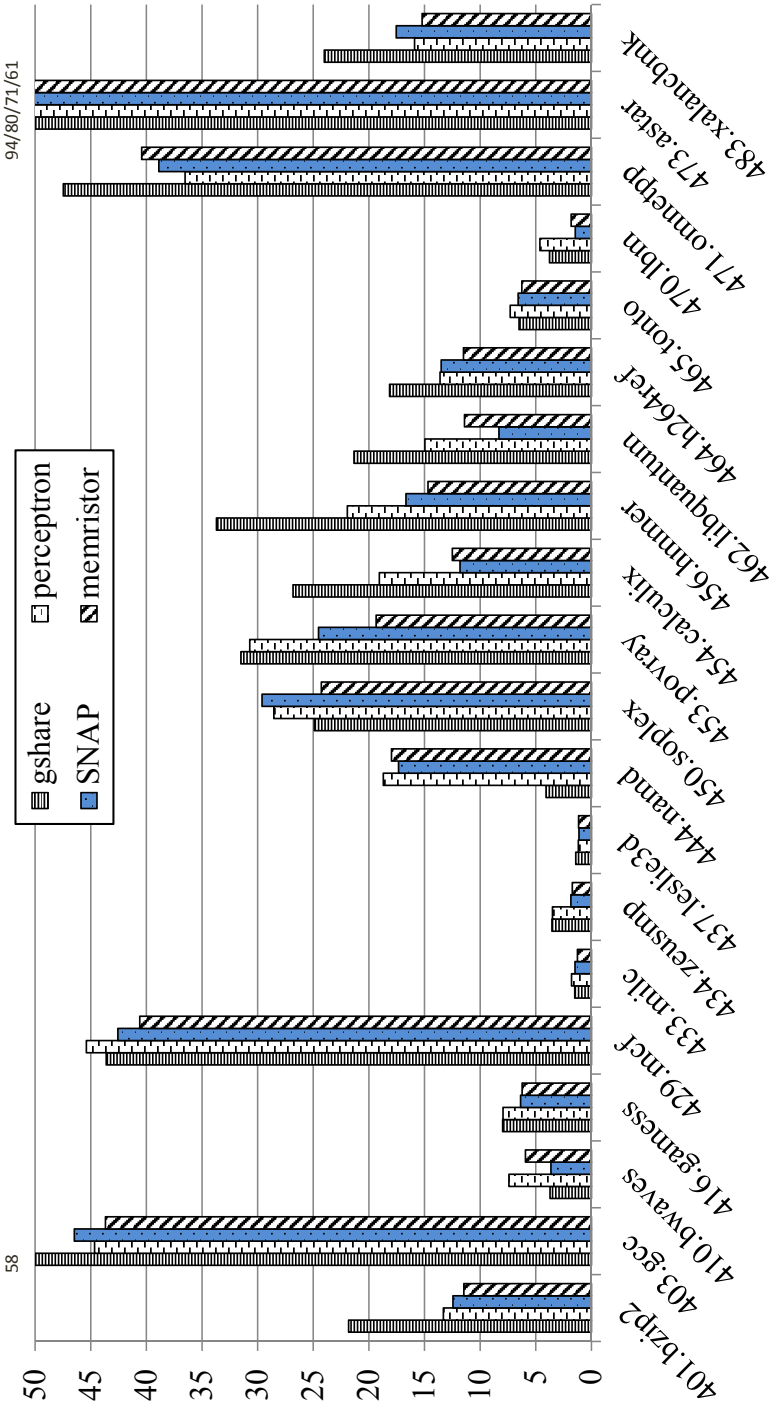


Fig. 3.7 Mispredictions per kilo-instruction (MPKI).

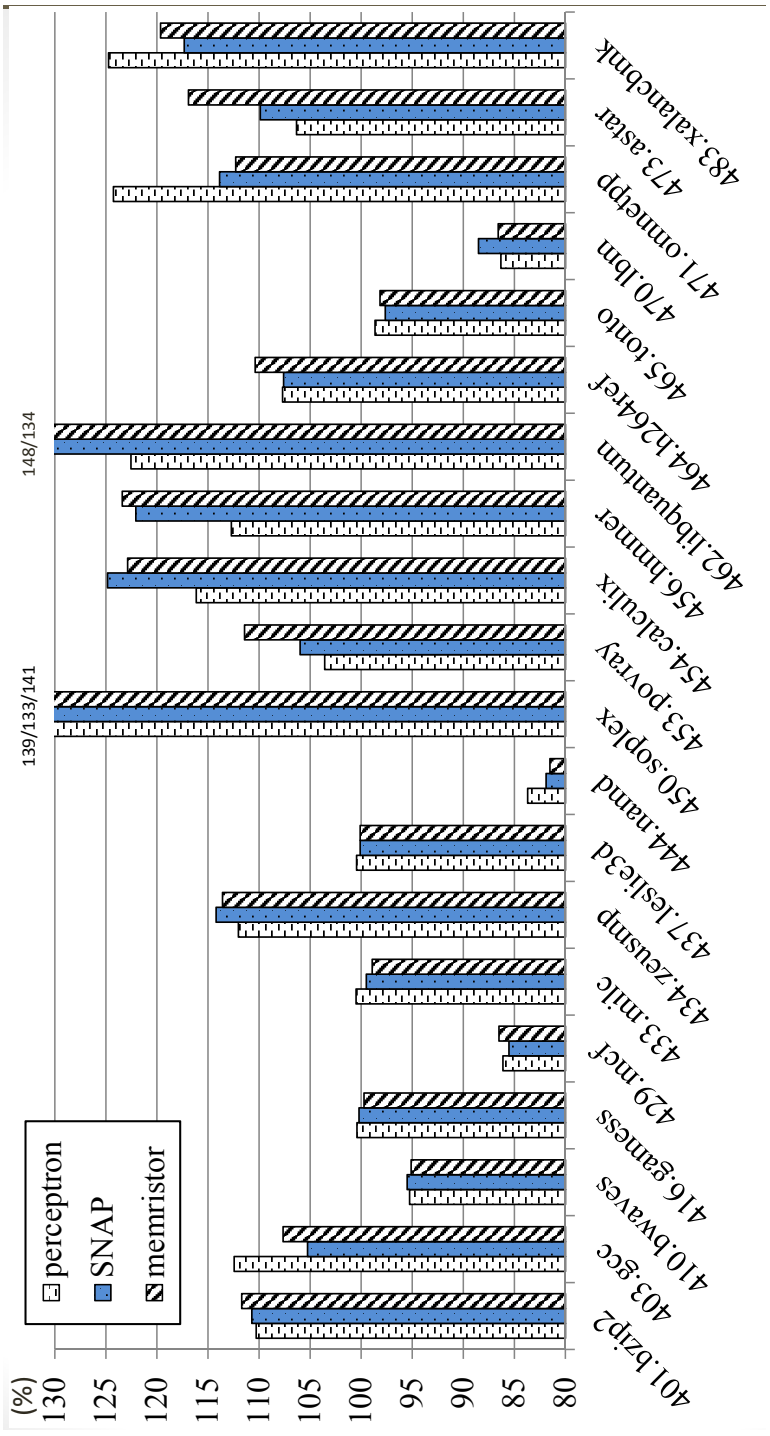


Fig. 3.8 Normalized IPC against gshare.

3.3.5 Prediction Confidence

Recall that the perceptron training process is invoked either after a misprediction has occurred, or the absolute value of the weighted sum is less than a threshold, indicating that the prediction is weak and the predictor need more trainings. However, MPKI does not provide information such as the confidence level of the predictor. In order to measure that, we shall analysis the training frequency of the neural predictors.

The data showed in Table 3.2 are the number of *Extra trainings per kilo-instruction* (ETKI) excluding those incurred by mispredictions. It demonstrates that how many predictions are indeed correct but are considered weak because of an under-trained predictor. As expected, the number of ETKI is a little more than MPKI since a neural-based predictor requires some amount of tuning efforts to switch to another branch behavior after a misprediction is encountered. However, for the digital perceptron predictor, the number of weak-but-correct predictions is much larger than the mispredictions in certain cases like `444.namd` and `471.omnetpp`.

Compared to perceptron, our memristor-based analog predictor significantly reduces the number of extra trainings while preserving a similar or even improving MPKI and IPC. Such an increased prediction confidence can be attributed to the inherent neural properties of memristors. Notice that the dynamics of memristor on its resistance change is a continuous, and highly nonlinear process rather than the simple discrete $+1$ or -1 arithmetic. In addition, the analog summation not only involves the memristor and resistor, but is also affected by other CMOS transistors. Even under a simplified model, the MLMCs together

	Perceptron	Memristor	Normalized Difference
401.bzip2	15.3	11.7	23.5%
403.gcc	76.2	31.8	58.3%
410.bwaves	8.9	6.5	26.4%
416.gamess	10.2	7.9	22.7%
429.mcf	52.5	34.4	34.6%
433.milc	2.4	1.4	44.2%
434.zeusmp	2.4	4.1	-72.4%
437.leslie3d	1.4	1.5	-2.9%
444.namd	73.8	12.5	83.1%
450.soplex	35.5	22.5	36.6%
453.povray	33.5	21.6	35.6%
454.calculix	16.1	16.2	-1.0%
456.hmmer	14.5	24.2	-67.1%
462.libquantum	12.1	14.3	-17.4%
464.h264ref	20.9	10.9	47.9%
465.tonto	11.4	4.5	60.7%
471.omnetpp	92.6	25.8	72.1%
473.astar	54.6	51.5	5.7%
483.xalanbmk	34.9	11.7	66.4%
Average	30.0	16.6	24.1%

Table 3.2 Extra trainings per kilo-instruction (ETKI).

form a much more complicated system with the analog output than a simple summation of weights in the digital domain, thereby behaving more closely to a real-life neural network.

3.4 Related Work

The *scaled neural analog predictor* (SNAP) [77] is the first neural-based branch predictor that aims to be both accurate and feasible. It uses analog computation techniques that are commonly found in the modeling of neural network to cal-

culate the compute-intensive dot-product. Such a mixed-signal design not only yields a manageable prediction timing, but also has a better energy-efficiency than the digital counterpart. However, SNAP still assumes the weight table to be stored in fast SRAMs which consume static energy.

A competitive memristor-based neural branch predictor was proposed in [34]. In that design, two memristors were used in both the left and right branches shown in Figure.3.5. Such a scheme aims at better sensing margins. However, in the case of neural branch predictor, reading out the exact value stored in the MLMCs is not necessary and only the aggregated output is concerned. A smaller sensing margin poses limited impact on the accuracy but consumes more energy. Furthermore, the author did not evaluate such a design at architecture level and failed to compare it with other analog schemes such as SNAP.

3.5 Summary

Neural branch predictors have been proven to perform very well in terms of accuracy. However, they face significant implementation challenges especially under the stringent demands of today's microprocessors. In this work we have proposed a more practical and energy-efficient neural predictor design based on a promising device, namely the memristor. It mitigates the compute-intensive parts of the neural prediction algorithm by utilizing analog computation techniques. The use of the memristor as both a storage component as well as a compute element achieved significant leakage energy savings while maintaining the same level of prediction accuracy and system performance. In addition, the inherent

property of the memristor increases prediction confidence with less predictor trainings, further reducing the dynamic energy consumed.

Our scheme can conceivably be applied in other neural-inspired predictors with additional modifications. Overcoming the latency issue involved during the prediction step by going analog enhances the feasibility of most neural predictors. More aggressive neural prediction algorithms such as [40] might now be practical. As the technologies continue to evolve, there seems to be a role for analog computing in processor architectures. The memristor, being CMOS compatible, opens up a number of applications [28, 44, 65] that are expensive to be implemented in a purely digital environment. This offers a promising direction in answering the energy challenge that comes with CMOS process scaling.

Chapter 4

Architecting STT-RAM as Energy-Efficient L1 Cache

In this chapter, we move down to the next level in the memory hierarchy: *first-level cache* (L1). At this level, performance is still important because most memory operations are read which stay on the critical path. In addition, it comes with an increasing requirement on capacity which is limited by the silicon area and power budget. Therefore, with the help of an emerging memory technology, namely the STT-RAM, we tried to build a more energy-efficient L1 cache in a larger capacity. Note that here we only deal with data cache, thus the term 'L1 cache' is referred to the first-level data cache in the subsequent sections unless stated individually. Figure 4.1 shows the scope of this chapter and the memory technologies involved.

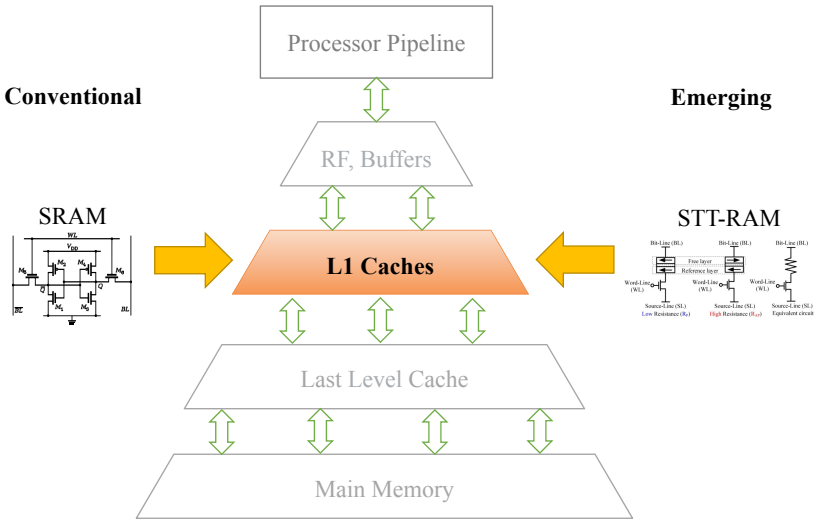
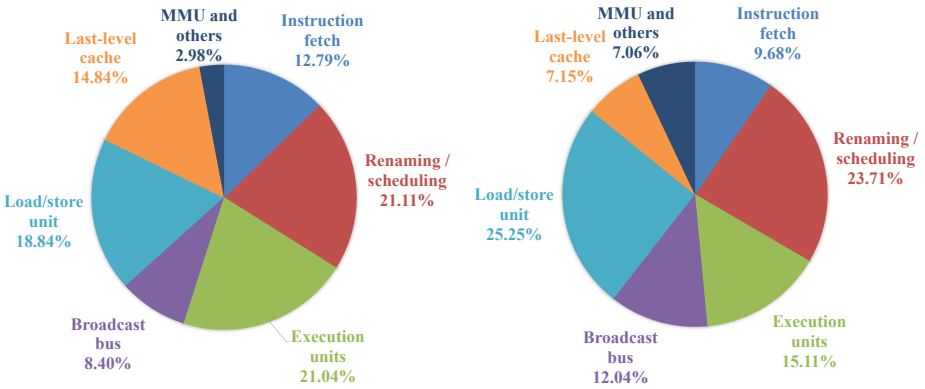


Fig. 4.1 Scope and memory technologies used in this chapter.

4.1 Motivation

Though the use of STT-RAM as on-chip caches has been widely studied [100, 93, 82, 98, 45, 37, 89], most of the works focus on lower-level caches because of 1) they occupy larger portions of the processor die; 2) the expensive write operation of STT-RAM contradicts to the high access frequency of first-level caches. However, these reasons do not necessary prohibit the use of STT-RAM in L1 cache. Figure 4.2 presents the runtime energy consumption breakdown for two benchmarks from the PARSEC suite. As we will discuss in later section, *fluidanimate* and *raytrace* represent a read-intensive and a write-intensive scenario respectively. In particular, the load/store unit consumes about $1/5$ to $1/4$ of the total energy and more than 80% of them are consumed by L1 data cache. The energy consumption per data block in L1 is indeed much higher than in lower level caches.



(a) fluidanimate (b) raytrace

Fig. 4.2 Energy consumption for each core component.

In addition, the high frequency of write in L1 does not make STT-RAM to be totally infeasible. There are several aspects of this issue. First of all, L1 cache is private, meaning that the number of conflict misses induced from different contexts are much less than those in lower-level shared caches. Thus, L1 cache typically has a higher hit rate with less block fills (considered as writes). Secondly, HPC applications in general contain more reads than writes (section 4.3.1) and with the help of *write buffering*, the actual number of writes to L1 cache are much less than reads. Thirdly, by relaxing the non-volatility of STT-RAM cell [75], we can reduce both the write latency and energy to some extent. Last but not the least, the cache coherence protocol provides some insights on the block behavior and we may further reduce the writes to STT-RAM by relocating some write-intensive blocks.

Based on these observations, the idea of *hybrid cache* has been proposed. It is an analogy to ARM's big.LITTLE architecture. The purpose is to put most of the data in a large STT-RAM partition in order to benefit from the low static power, and to use an additional SRAM partition to filter out the expensive writes.

As mentioned, because of the R/W asymmetry, the size of the SRAM partition can be much smaller than STT-RAM with manageable energy and performance overhead. Section 4.1 will describe our proposal in more details.

4.1.1 Cache Coherence

Issues of private cache

In a shared-memory multiprocessor system, each processor node is typically designed to have its own write-back private (local) caches for performance reason. Thus, any data block may end up stored in multiple private caches and subjected to local changes without notification. Figure 4.3 illustrates a scenarios where data block A is presented in the local cache of processor P0 and P1, and data block B is shared between P0 and P2. The lower-level memory (shared cache) contains both blocks and other data that may not presented in upper level caches. A sequence of operations issued by each processor is given in the right side. We assume each private cache fetches data from lower-level shared cache and returns an updated version when the block is evicted. Without a communication mechanism, P1 will be unaware of the changes to A by P0 and end up reading the expired data from its local copy. Furthermore, as both P0 and P2 have modified block B, three different versions of B are existed together in the system (local copies of P0 and P2, and the old copy in lower-level cache). Depending on when the read operations arrive in lower-level cache, the value of B which P0 reads may not be consistent with the value that P1 reads, given these two reads happen at the exact moment.

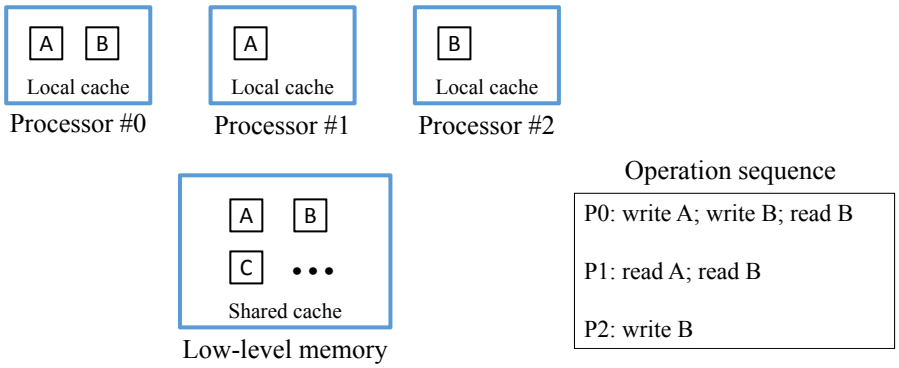


Fig. 4.3 A scenario of private-shared caches.

Coherence protocol

In order to resolve such conflicts and maintain the consistency between the shared data copies, *cache coherence protocol* has been developed. It defines how the memory operations are propagated through the system so as everyone sees the same order of changes. Two most common approaches ensuring cache coherence are *directory-based protocol* and *snooping-based protocol*. In directory-based protocol, the shared data are placed in a central directory. It maintains the sharers' information and keeps track of the most recent data copy. Any modification to shared data needs the permission from the central directory. The directory will informs other sharers about the changes and depending on the implementation, other caches may either invalidates its old copy or simply update it.

As an alternative, snooping-based protocol does not rely on a central directory. Instead, each cache maintains the state information (for example, shared or exclusive) for each data block. The local caches monitors each other's activities through a common data bus and responses when necessary. One of the widely used snooping-based protocol is MESI [63]. MESI models each data block as a finite-state machine. Based on a block's MESI state, certain memory requests are

required to snoop the bus first and broadcast themselves. Any other cache holding the same copy must response to such requests by changing its own copy's state or writing back its copy to lower-level memory. Any further snooping request needs to wait until the bus is released.

In additional to the original version developed by the University of Illinois, several MESI variants have then be introduced with different design focuses. In particular, MOESI extended the original MESI with the capability of dirty-sharing by adding an auxiliary state and has been used in AMD's processor family [5].

In MOESI, each cache block can be in one of the five following states:

- **Modified (M)** - The cache block is an exclusive copy that only presents in the current cache. In addition, it is dirty and must be written back to lower-level memory before eviction.

- **Owned (O)** - The cache block must be written back to lower-level memory before eviction. It may or may not present in other caches as a shared block. Therefore, any modification to it must be broadcast.

- **Exclusive (E)** - The cache block is an exclusive copy that only presents in the current cache. However, it is a clean copy and can be silently evicted without writing back to memory.

- **Shared (S)** - The cache block is a clean copy but might present in other caches at the same time. It shall update the data when receive a broadcast from the owned cache. Silent eviction is allowed.

- **Invalid (I)** - The cache block is not in use.

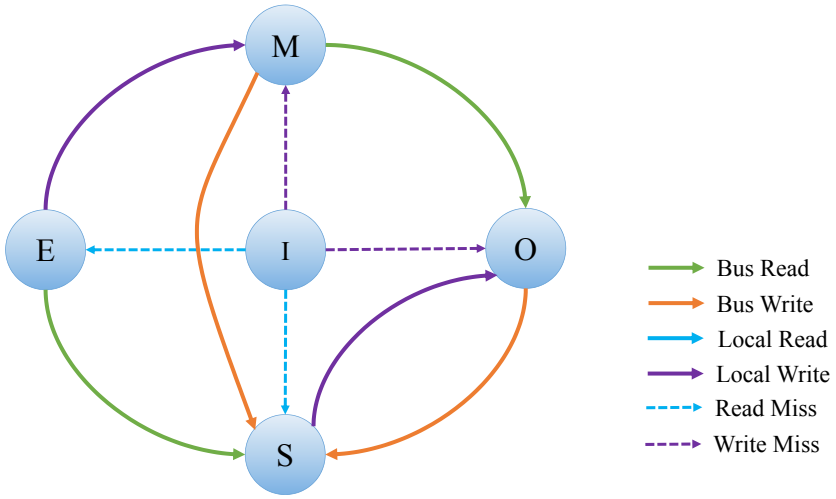


Fig. 4.4 MOESI coherence protocol.

		Other Caches				
		M	**O**	**E**	**S**	**I**
Own Cache	**M**	N	N	N	N	Y
	O	N	N	N	Y	Y
	E	N	N	N	N	Y
	S	N	Y	N	Y	Y
	I	Y	Y	Y	Y	Y

Table 4.1 Permitted co-exist state relation in MOESI.

For each data block, its coherent state changes based on two types of events: *remote (bus)* operations issued by other nodes and *local (processor)* issued by own processor. The state transition diagram with major trigger events is shown in Figure 4.4. The relation of permitted co-exist state is shown in Table 4.1 where 'Y' and 'N' denotes possible and impossible respectively.

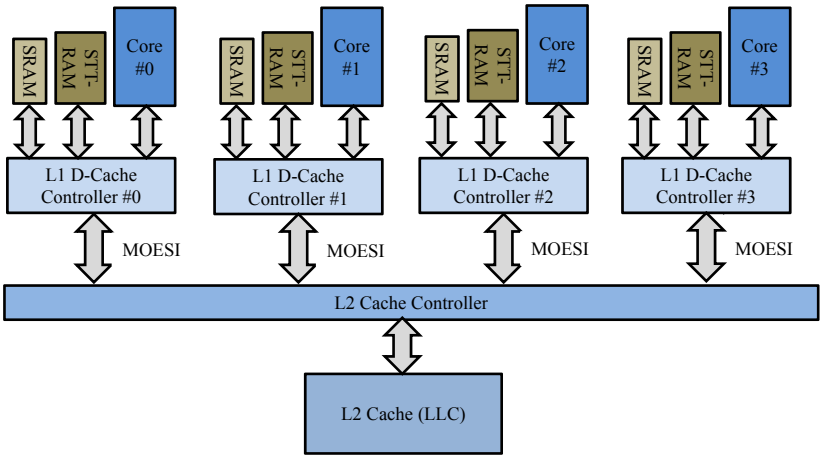


Fig. 4.5 The hybrid cache hierarchy.

4.2 A Hybrid L1 Cache Architecture

Figure 4.5 shows the overview of our proposed hybrid L1 cache architecture for a modern multi-core processor. Each of the processor core has a dedicated L1 cache comprising of a SRAM and a STT-RAM partition while all the cores are sharing a unified last-level cache (L2). The L1 cache controller is responsible for handling cache access requests (either from its owned processor core or the common bus), and migrating cache blocks between the two partitions based on the information provided by the coherence protocol.

4.2.1 Cache Block Management (Naïve Solution)

The main purpose of a block management policy is to relieve the write pressure on STT-RAM partition. However, the difficulty lies in predicting a cache block's accessing behavior. A naïve policy would be allocating all read-miss blocks to STT-RAM partition and the rest to SRAM partition during cache fills. This method assumes for every cache block, the first operation to it would be the

dominant one until it gets evicted. However, there is no reason that this is so. For example, during an application's warming up phase, cache behavior is generally unstable, and such an assumption is most likely invalid.

In order to implement the naïve management policy, additional circuitries are added to the conventional cache architecture. Figure 4.6 illustrates how a read operation is performed. Since the sizes of the STT-RAM and SRAM partition are different, the memory address is firstly decomposed into two pairs of tag and index (Step 1). Then both partitions are accessed simultaneously, and at most one partition would produce a cache hit (Step 2). Upon the cache hit, the corresponding data is sent via a 2-to-1 multiplexer (MUX) selected by the STT-RAM (or SRAM) hit signal to the processor core (Step 3). In any case, the extra delay caused by the MUX is negligible since the accesses to the tag arrays in step 2 are done in parallel.

The type of memory operation determines which partition would handle the cache miss. Figure 4.7 illustrates this procedure. A demultiplexer (DEMUX) is added at the top to direct the input data from lower level memory to the corresponding cache partition. Note that dealing with a cache miss potentially involves evicting an existing least-recently-used (LRU) cache block. Since STT-RAM has a longer write latency than SRAM, read-misses require more time to handle than write-misses under write-allocation policy.

4.2.2 Cache Block Management (Immediate Migration)

The naïve block management policy assumes static or fixed allocation where each cache blocks stay in only one partition during its lifetime. Such a simple

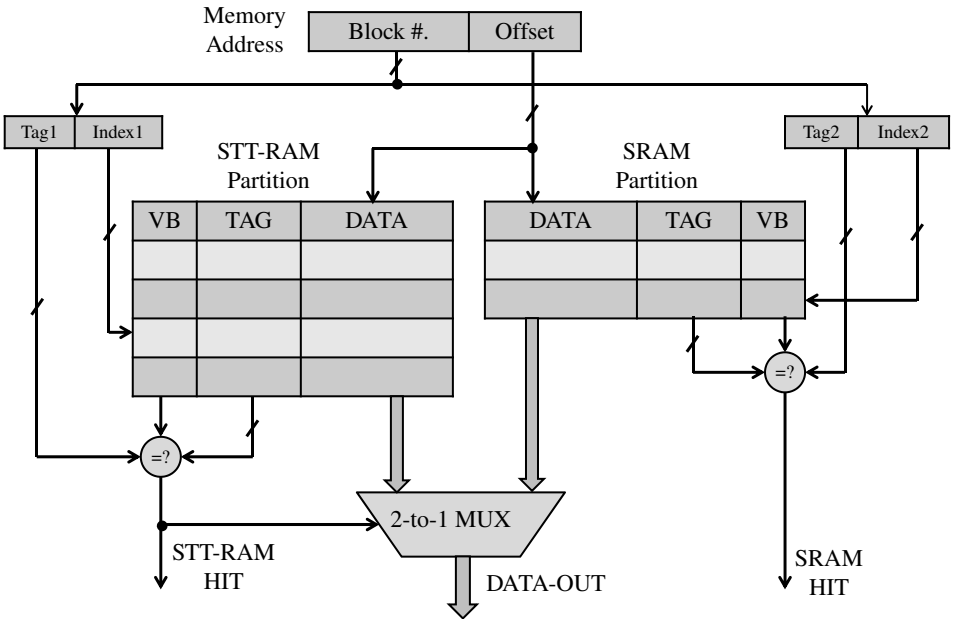


Fig. 4.6 Data path of read operation for the hybrid cache.

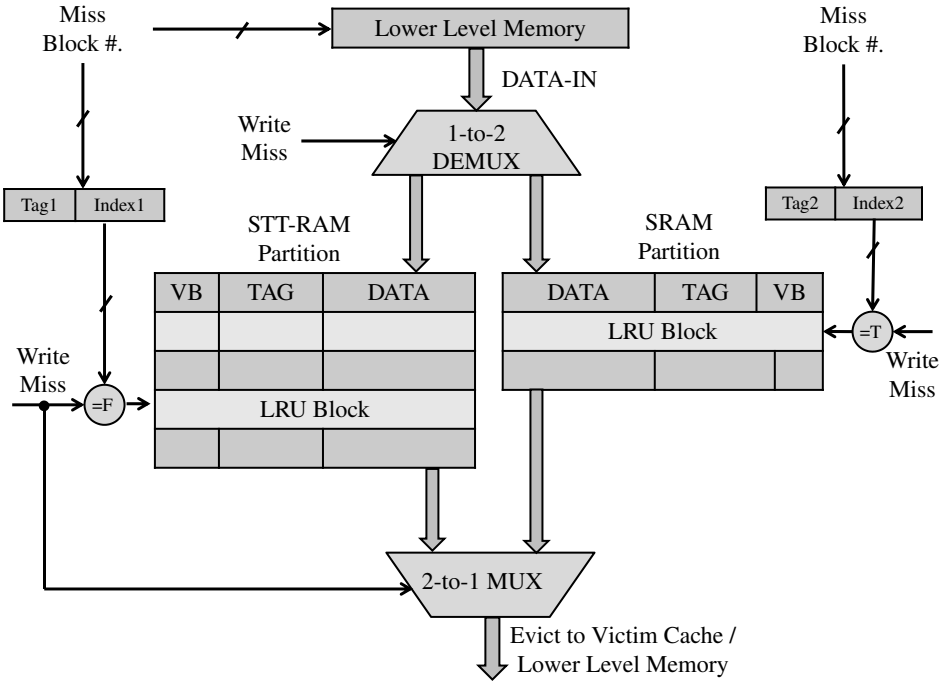


Fig. 4.7 Miss handling for the hybrid cache.

policy fails to consider a cache block's dynamic access behaviors and so thus a more intelligent policy shall reallocate the cache blocks based on their periodical behaviors. Figure 4.8 shows a block's transition state and movement under a proposed dynamic policy.

While the naïve solution is based entirely on temporal locality, *immediate migration policy* (IMP) takes things a step further. In the MOESI protocol, if a cache block is in `MODIFIED` (`M`) or `OWNED` (`O`) state, it must has been written to before and more importantly, it probably will be written to again in the near future. Therefore, such blocks most likely would receive larger amount of writes compared to those in other states and hence would benefit from staying in the SRAM partition of the cache. On the other side, `EXCLUSIVE` (`E`) or `SHARED` (`S`) states indicate that the block is a read-only copy for now, and is likely to stay as it is. Hence, they better to reside in the STT-RAM partition.

While a remote (snooping) memory write operation that hits on an `O` state block in the SRAM partition, an ownership transfer is required. As such, it changes the local block state to `S` and cause the corresponding block to be migrated to STT-RAM partition. On the other side, a local write hit on the STT-RAM partition makes the block dirty, and necessitate its transfer to the SRAM partition of the cache under its new `M` or `O` state. This process requires no extra information since all migrations are determined by the transition of coherent states. Besides, the way of handling cache misses remain as the same as the naïve solution.

Note that the order of handling the actual memory request and migrating the affected cache blocks can potentially lead to differences in performance and energy saving, but we found that such variations are generally small, and can be

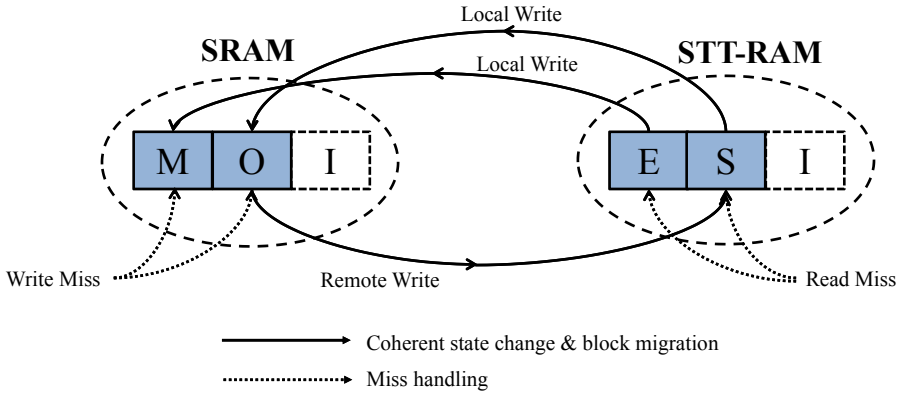


Fig. 4.8 Immediate migration transition diagram.

safely ignored. Throughout this work, we assume the block migration completes before serving the memory request.

4.2.3 Cache Block Management (Delayed Migration)

Under certain circumstances, IMP could be too aggressive since a cache block is migrated immediately after a coherent state change. For example, some of the `M` state blocks are essentially read-intensive and would benefit more from staying in STT-RAM; while some `S` state blocks receive more broadcast updates from the owner cache than local reads, making the migration to STT-RAM unnecessary. As an alternative, the *delayed migration policy* (DMP) offers a way to postpone the migration until a possible better timing. Figure 4.9 illustrates the migrating decision making process.

In DMP, only when two consecutive operations satisfying the migrating condition are encountered will the block migration be initiated. In particular, two consecutive local writes must have been encountered for a block to be migrated from STT-RAM to SRAM partition, regardless of the coherent state changes.

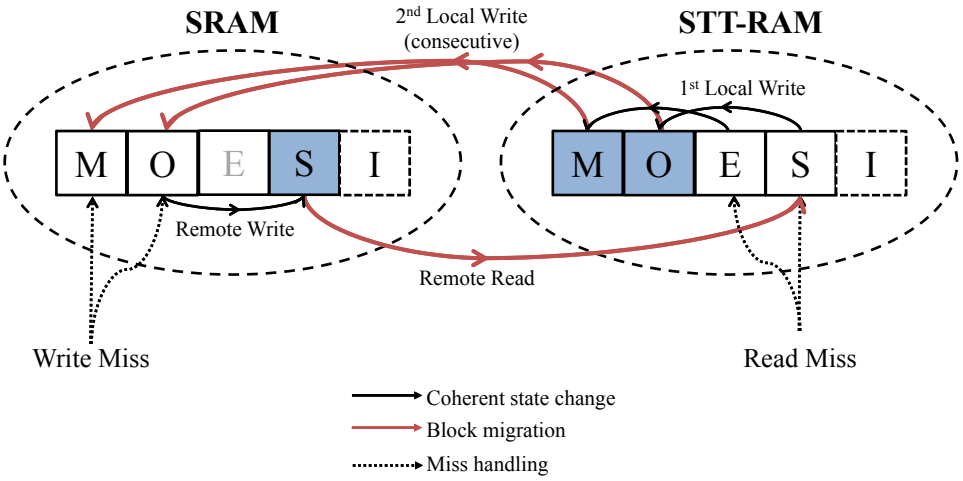


Fig. 4.9 Delayed migration state transition diagram.

As a necessary condition, such a block should already be in either `M` or `O` state. Similarly, two remote reads with no intervening writes will migrate the block in either `S` or `O` from SRAM to STT-RAM partition. Under such a migration policy, `E` state block can only exist in STT-RAM partition because the transition is only caused by read-misses and any write to it will lead to a state change.

In order to track the migration state, an extra *transfer determine* bit (TD-bit) is appended to each cache block. The set/reset policy are stated as follows. All newly-allocated blocks will have their TD-bit reset, regardless of the partition. For a `O` or `S` state block resided in the SRAM partition, a remote read sets the TD-bit while any write (including those broadcast updates from the owner) will have it reset. Similarly, on the STT-RAM side, only local writes set the TD-bit and any other reads will reset it. Such a scheme also reduce the thrashing effect where a block keep migrating between the two partitions due to frequent state changes. It is often observed during synchronization between different threads.

Processor	
Core frequency	3GHz, 8 out-of-order cores
Dispatch width/window	4 / 128
Outstanding loads/stores	32
Branch predictor	gshare, 16 history bits
ITLB/DTLB	64 / 64
Private L1 Cache	
Instruction (SRAM)	32KB, 4-way, 32-byte
Data (SRAM)	4-64KB, 4-way, 64-byte
Data (STT-RAM)	64-192KB, 4-way, 64-byte
Coherent Protocol	MOESI
Shared L2 Cache (LLC) and Main Memory	
L2 (SRAM)	4MB, 8-way, 64-byte, 15-cycle R/W
Memory (DRAM)	64 banks, 150-cycle latency

Table 4.2 Simulation platform.

4.3 Evaluation

4.3.1 Experiment Setup

Simulation platform

To evaluate our design, we modified the *sniper* multi-core simulator [12] to model a conventional x86 processor with a two-level cache hierarchy as shown in Figure 4.5. Sniper is a high-speed and yet accurate x86 multi-core simulator built on the *pin* tool [70]. It adopts analytical core models to trade off simulation speed for internal core structure details [13]. The list of simulation parameters is given in Table 4.2.

Benchmark characteristics

Twelve diverse multi-threaded workloads from PARSEC [10] benchmark suite were simulated. We divided them into three categories: *read-intensive*, whose

Workloads	Application Domain	Reads (%)	Writes (%)	R/W Behavior
blackscholes	Financial Analysis	65.65	34.35	write-intensive
bodytrack	Computer Vision	80.30	19.70	read-intensive
canneal	Engineering	66.67	33.33	write-intensive
dedup	Enterprise Storage	72.32	27.68	balanced
facesim	Animation	70.08	29.92	balanced
ferret	Similarity Search	79.36	20.64	read-intensive
fluidanimate	Animation	80.65	19.35	read-intensive
raytrace	Rendering	54.40	45.60	write-intensive
streamcluster	Data Mining	86.73	13.27	read-intensive
swaptions	Financial Analysis	76.7	23.3	balanced
vips	Media Processing	80.51	19.49	read-intensive
x264	Media Processing	76.21	23.79	balanced

Table 4.3 Memory operation breakdowns in PARSEC workloads.

read-to-write ratio is much larger than 3; *balanced*, whose read-to-write ratio is around 2 to 3; and *write-intensive*, whose read-to-write ratio is smaller than 2. The detailed description of each benchmark along with the breakdowns of their memory operations are shown in Table 4.3.

Hybrid cache configuration

Both of the SRAM and STT-RAM cache energy and latency numbers were generated using NVSim [26] with a 32nm technology node assumed. The baseline case is set to a 64KB pure SRAM cache. While exploring other possible cache size configurations, we take care not to exceed the silicon area of the baseline. The feasible configurations are given in Table 4.4 and the energy and latency numbers are in Table 4.5.

Note that STT-RAM is a storage-class memory that can retain data for more than 10 years at room temperature. However, to achieve this requires a programming time that is 10× that of SRAM [82]. One can shorten this significantly by

Hybrid	SRAM	STT-RAM
No	64KB	N.A.
No	N.A.	192KB
Yes	4KB	128KB
Yes	8KB	64KB

Table 4.4 Feasible configurations.

	SRAM (KB)			STT-RAM (KB)		
	4	**8**	**64**	**64**	**128**	**192**
Read / Write Latency (cycle)	3 / 3	3 / 3	4 / 4	3 / 9	4 / 10	4 / 11
Dynamic Read Energy (nJ)	0.055	0.063	0.095	0.055	0.064	0.085
Dynamic Write Energy (nJ)	0.053	0.061	0.093	0.279	0.3	0.35
Leakage Power (mW)	15	28.7	150	8.7	12.1	25

Table 4.5 Performance and energy parameters.

relaxing the non-volatility requirement of STT-RAM [75]. As such, we added a conventional DRAM-style refresh mechanism to maintain the data integrity.

Energy modeling

The total energy consumption for L1 cache is modeled by the sum of three components: *leakage energy*, *refresh energy* and *dynamic energy*. Besides the usual runtime read/write energy, dynamic energy in our context also includes the energy for probing both cache partition (regardless of hit status), and the *migration energy* when one of the block migration policies is applied. The broadcast write energy generated from the updates of O-state blocks to other corresponding S-state blocks have been included as well.

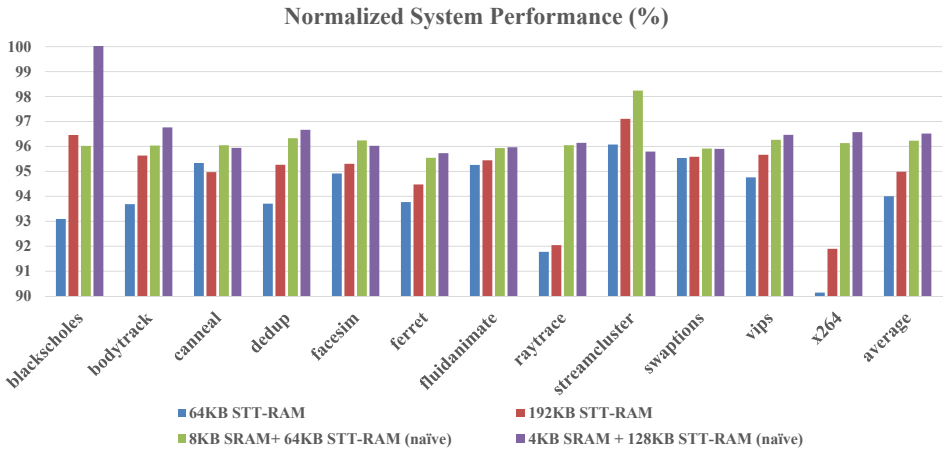


Fig. 4.10 Normalized system performance for pure STT-RAM and hybrid cache without block migration.

4.3.2 Static Block Management

We begin by first considering the impact of directly using a pure STT-RAM setting to replace a SRAM-based L1 cache. NVSim simulation shows that 3× of the capacity can be achieved for STT-RAM within the silicon area of 64KB SRAM. Therefore, we simulated two pure STT-RAM L1 caches, one with a 64KB STT-RAM for a *same-capacity* comparison, and another with a 192KB STT-RAM for a *same-area* comparison with the baseline. Figure 4.10 shows the normalized system performance of these two scenarios, which is measured through *instructions-per-cycle* (IPC).

As expected, all the benchmarks running under a same capacity STT-RAM cache suffered performance degradation due to the longer write latency. On average, system performance deteriorated by about 6%. When the cache capacity is enlarged to 192KB, performance generally improved, but the gain is not significant. Overall, the system performance only improved by 1% even with a cache that is three times larger. Interestingly, *canneal* performed slightly worse

	No Migration		Immediate		Delayed		Queue-based	
	Reads (%) STT-RAM	Writes (%) SRAM	Reads (%) STT-RAM	Writes (%) SRAM	Reads (%) STT-RAM	Writes (%) SRAM	Reads (%) STT-RAM	Writes (%) SRAM
blackscholes	65.3	100.0	65.8	100.0	65.3	100.0	63.5	100.0
bodytrack	72.8	88.1	69.9	99.6	70.0	98.9	76.7	87.9
canneal	91.2	54.2	80.5	99.2	84.1	94.4	84.3	94.0
dedup	84.9	21.6	38.0	95.7	53.0	79.8	78.0	46.6
facesim	68.9	50.5	47.9	98.6	51.5	92.5	62.6	66.4
ferret	73.7	57.9	45.7	96.9	52.8	82.1	72.0	69.5
fluidanimate	88.2	40.8	75.2	98.9	76.5	91.2	84.0	72.2
raytrace	31.9	94.8	27.1	99.8	26.5	99.7	38.9	90.1
streamcluster	73.5	76.1	62.2	98.8	71.1	83.7	83.1	8.7
swaptions	91.1	31.0	62.1	97.7	65.1	84.8	84.6	50.4
vips	84.5	92.2	69.2	99.9	69.2	99.7	93.8	93.3
x264	65.8	74.4	51.5	99.0	51.8	95.2	65.1	86.4
Average	74.3	65.1	57.9	98.7	61.4	91.8	73.9	72.1

Fig. 4.11 R/W distribution for a 4KB SRAM + 128KB STT-RAM hybrid cache.

in a larger cache setting. This is because the marginal performance benefited from a lower miss rate is unable to compensate the latency increased from a larger capacity. In summary, a direct replacement of SRAM by STT-RAM is not very attractive due to the degradation of system performance.

Most of the benchmarks received performance increase after adding a fast SRAM partition for the handling of write-misses. Some of them can even match the performance of the baseline. The difference between the two cache configurations is merely 0.2%, indicating that the additional misses caused by varying size of both partitions are not performance critical under the naïve policy. On average, the hybrid cache without block migration still suffers about 3.4% performance loss in the best case.

4.3.3 Dynamic Block Management

To evaluate the effectiveness of our migration proposals, we first calculate the read-write distribution for each cache partition. Three dynamic block management policies have been considered: *immediate migration*, *delayed migration* and *queue-based migration* proposed in [82]. Table 4.11 shows the percentages of reads in STT-RAM partition and writes in SRAM partition in a 4KB SRAM + 128KB STT-RAM hybrid cache setting. Under the naïve policy where no block migration is taking place, the additional SRAM partition can filter out $2/3$ of the total writes only, leaving $1/3$ writes to the STT-RAM partition. When the immediate migration policy is adopted, most of the writes are forced to operate on the SRAM partition, except for those broadcast updates from O-state blocks to S-state blocks. At the same time, it also reduced the reads to STT-RAM partition by 16.4%. For the delayed migration policy, though the number of block migrations is reduced, it still prevented a high amount of writes (more than 90%) to the STT-RAM partition with an increased number of reads. For comparison, queue-based migration resulted in a higher amount of reads to STT-RAM, but it also suffered from a larger number of writes to STT-RAM partition as well.

Figure 4.12 shows the normalized system performance. We have tested two hybrid cache configurations: one with 8KB SRAM and 64KB STT-RAM and the other with 4KB SRAM and 128KB STT-RAM. It turns out that most benchmarks are not sensitive to the size of SRAM partition and are more favorable to a larger STT-RAM partition when any of the migration policy is adopted. On the other hand, except for `dedup` and `facesim`, the performance difference between immediate and delayed migration policies in a larger hybrid cache setting is not

significant. On average, all dynamic block migration policies performed better than the static naïve policy and the queue-based migration policy.

On the energy side, Figure 4.13 shows the normalized energy consumption. Regardless of the migration policies, all hybrid caches achieved substantial energy savings. In particular, more than 40% of energy saving has been recorded in `canneal`. On average, the naïve block management scheme reduced the energy consumption by around 20% while our migration schemes saved additional 15% compared to the baseline. Although the queue-based migration did maintain a comparable performance, its energy consumption is on a par with the naïve policy only.

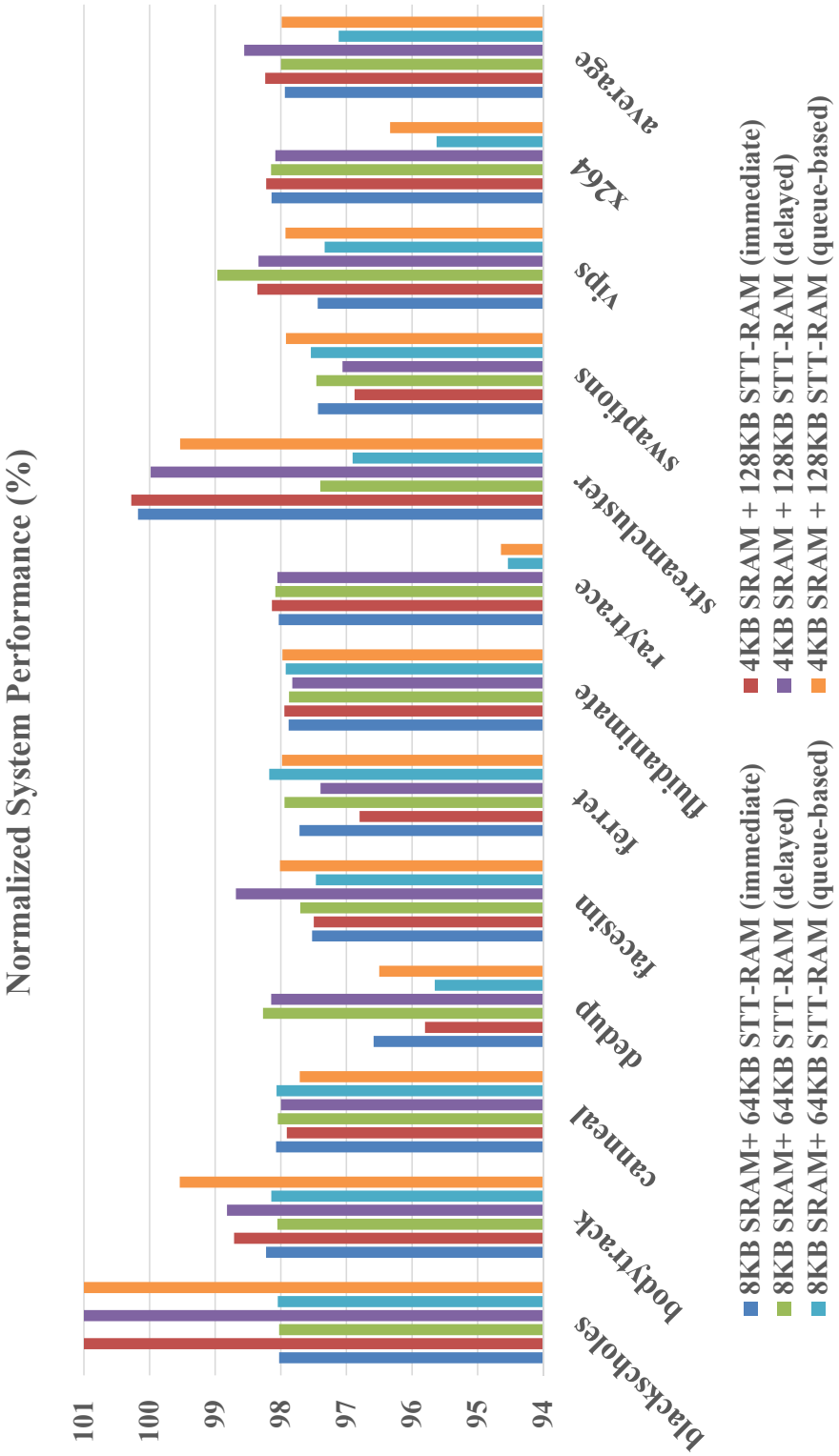


Fig. 4.12 Normalized system performance for hybrid cache with different migration policies.

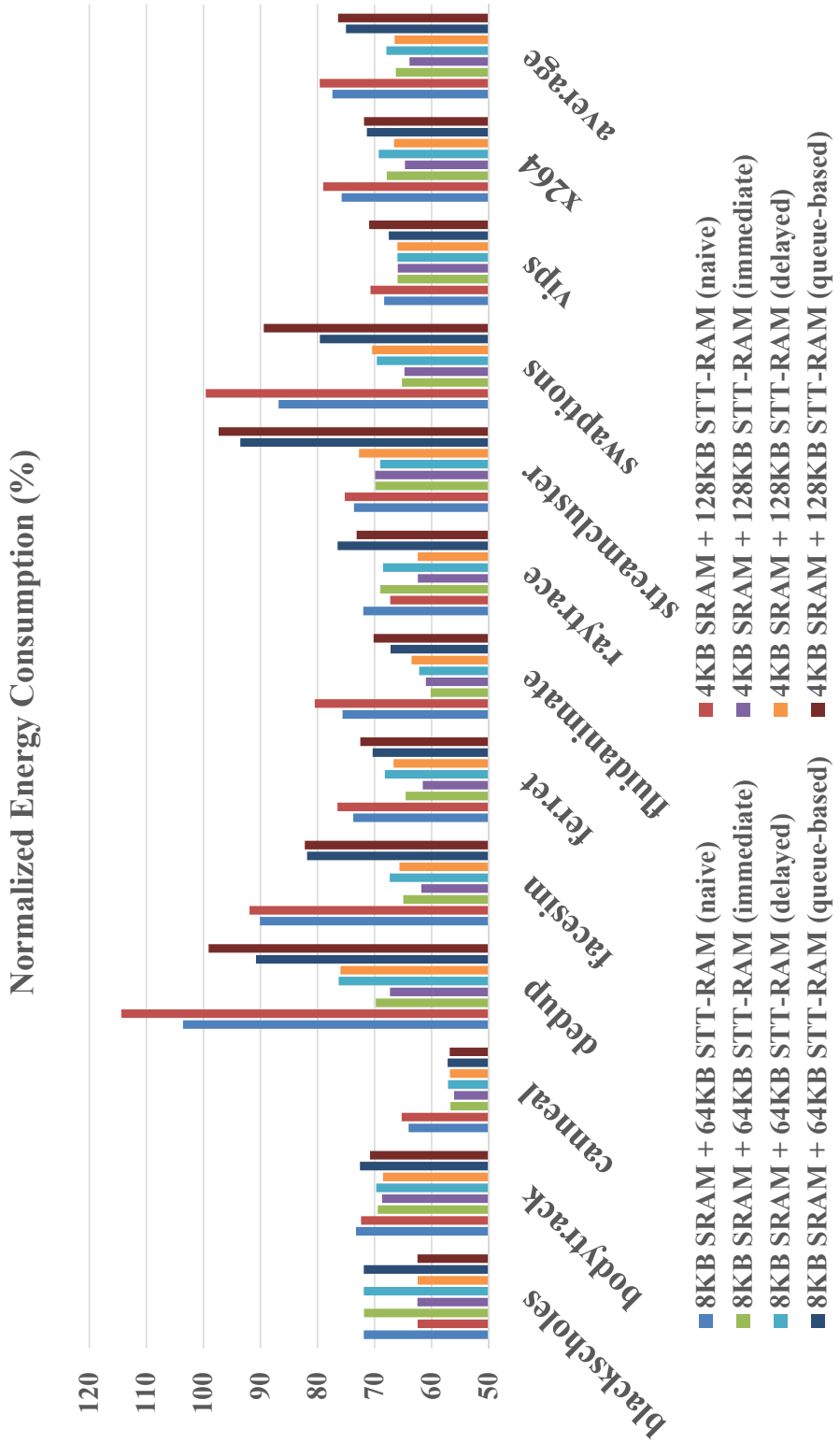


Fig. 4.13 Normalized total energy consumption for hybrid cache with different migration policies.

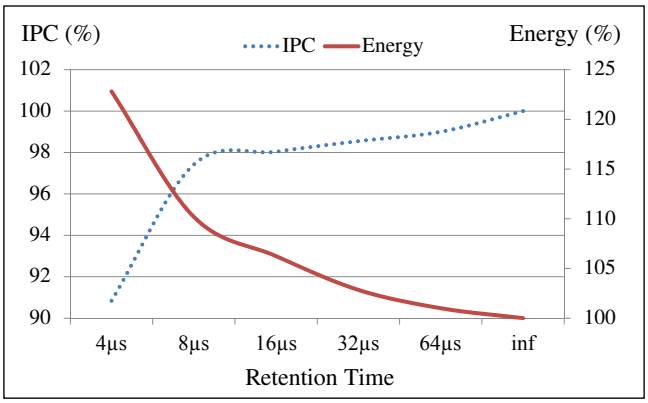


Fig. 4.14 System performance and energy consumption for various STT-RAM retention times.

4.3.4 Impact of Retention Time

The retention time of STT-RAM cells is related to the thermal barrier Δ of an MTJ, which can be expressed as $t = C \times e^{k\Delta}$ where t is the retention time, and C and k are the fitting constants [82]. Any variations in the planar area and the thickness of the MTJ affect the thermal barrier, and thus impact the retention time. However, given a particular set of read/write latency and the cache size, the lowest possible retention time under a DRAM-style refresh scheme is bounded by $\#cache\,blocks \times (read\,latency + write\,latency) \times cycle\,time$.

Figure 4.14 shows the IPC and energy consumption under various retention time values for a 8KB SRAM + 64KB STT-RAM hybrid cache with immediate migration policy. The data are normalized to the perfect case when the refresh is completely eliminated (inf). In general, a lower retention time causes more refresh conflicts and thus increases the latency for memory requests. It also consumes more energy due to a more frequent refresh schedule, but the impact become marginal while the refresh period is larger than $32\mu s$. We demonstrate these results is because STT-RAM are not yet in commercial deployment and the

manufacturing parameters are not fully known. An extreme short retention period can seriously downgrade performance and lead to a higher energy consumption.

4.4 STT-RAM Endurance Study

Although the *write endurance* issue for STT-RAM can be mitigated by using compiler techniques [51], architecture solutions have the advantage of being application-transparent and thus are more flexible in actual deployment. Prior work only considered the write endurance of STT-RAM in the context of last level caches [18]. However, the issue is even more pressing when STT-RAM technology is deployed in L1 caches as they have much more write activities than the last level caches. Although a prediction of 10^{15} programming cycles [84] is often cited as the write endurance for STT-RAM, real experiments thus far have showed that the number is less than 10^{13} write cycles [25]. This would be a severe constraint on any pure STT-RAM L1 solution [82], and is another key motivation for our hybrid design.

Figure 4.15 shows the average number of *writes per cycle* (WPC) to the STT-RAM partition. Among all the benchmarks, `facesim` has the highest average writes per cycle. Consider the case when the writes are perfectly distributed to all cache blocks in `facesim` for a pure STT-RAM design, each block would need to stand 2.4×10^5 writes per second. Under a conservative estimate, a cache block will last about 1.3 years on a 3 GHz processor.

However, further analysis shows that writes are not at all evenly distributed. This is true within a single cache as well as across different private caches. We observed that some blocks are seldom used while several other blocks suffer from

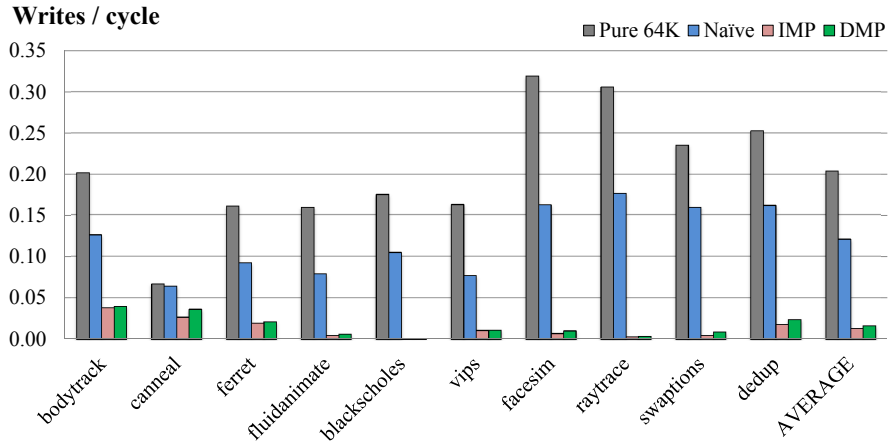


Fig. 4.15 Average number of STT-RAM writes occurs in each CPU cycle.

a huge amount of writes. For example, about 15% writes target at a particular block in `facesim`, while the corresponding cache partition receives nearly 50% writes among the four private caches. In another words, assuming a 3 GHz clock and a write endurance of 10^{13} write cycles, that STT-RAM block may fail within half an hour if no proper measure is taken. Due to the low-latency requirement and high access frequency of L1 caches, existing wear-leveling methods for last level caches [18] are not feasible.

Table 4.6 lists the average and worst case lifespan of the STT-RAM cache for `facesim` under the write endurance assumption of 10^{13} programming cycles. The “average partition" column assumes that the writes are perfectly distributed to all of the private caches while the “worst partition" only assumes the writes are evenly distributed within each private cache partition. In the “worst block" case, the actual number of writes per cache block is computed. Compared to a pure 64KB STT-RAM solution, the lifespan of the most write-intensive cache block in a hybrid configuration with the DT policy is increased by up to 150×, while the worst cache partition lifespan is increased by 2333%. Although DMP

	Average partition	Worst partition	Worst block
Pure STT-RAM	1.3 years	0.3 years	$<$ 22 mins
Hybrid Naïve	3.5 years	1.0 year	0.9 hr
Hybrid IMP	41.2 years	6.9 years	51.6 hrs
Hybrid DMP	32.9 years	7.0 years	54.3 hrs

Table 4.6 STT-RAM cache lifespan estimation for `facesim`.

performed slightly better than IMP in the worst case endurance, it suffered from a lower average lifespan because the blocks in STT-RAM received more writes due to a delayed migration policy. Note that the naïve solution for hybrid cache is still insufficient for actual deployment.

In practice, caches would come with redundancy or error correction code to improve reliability. Also, it is very unlikely to see such a sustained high write frequency on a particular block especially for personal workload. Thus the worst case block lifespan should be much longer. Nonetheless, the risk exists, especially for the current state-of-the-art in STT-RAM technology. As a by-product, our proposed hybrid architecture can significantly reduce this risk.

4.5 Related Work

Choosing STT-RAM as a replacement to SRAM in on-chip cache design is not brand-new idea. In the past, most of the works avoid deploying STT-RAM in L1 cache due to the prolong write operations. For example, Jog et al. [45] deployed pure STT-RAM in L2 cache with an application-driven retention time study. Smullen et al. [75] proposed a hybrid cache design with entirely SRAM-based L1, and STT-RAM-based shared L2/L3 cache at a level of relaxed non-volatility to accelerate write operation. Jiang et al. [37] used STT-RAM as the last level

cache under embedded settings with SRAM-based private L1 caches. In addition, *early write termination* has been proposed as a device optimization technique for STT-RAM write energy reduction [100].

One of the few STT-RAM-based L1 cache architectures was proposed by Sun et al. [82]. Their design was built using several MTJ designs with various levels of retention time. Additional SRAM buffers and detection logics were added to reduce the penalties from STT-RAM writes and refreshes. However, the overhead involved in the counter-based retention monitor might cause more performance impact on the L1 cache than using a simpler DRAM-style refresh.

4.6 Summary

In this work, we proposed a hybrid L1 cache architecture uses both conventional SRAM as well as the new STT-RAM technology. The larger STT-RAM partition of the cache allows for higher capacity with low energy consumption. The smaller SRAM partition filters out most of the write operations to the cache by exploiting the MESI cache coherence protocol. This significantly mitigates the impact of the STT-RAM's high write latency on IPC. A scheme to dynamically transfer cache block between the two portions of the cache was presented. The proposed hybrid architecture significantly reduces overall energy consumption while maintains a reasonable level of performance. Experiments showed that our proposed approach can achieve either more than 40% energy saving with less than 0.9% decrease in IPC or 0.1% improvement on IPC with nearly 30% energy saving when compared to a pure SRAM-based design.

As a new technology, STT-RAM faces certain reliability risks. In particular, the measured write endurance of STT-RAM cells reported is orders of magnitude below the L1 cache requirement. In particular, L1 writes are extremely skewed, and given the critical impact L1 access time has on performance, earlier wear leveling proposals for STT-RAM last level caches simply will not do. In this work, we investigated this issue, and showed that our proposed hybrid scheme can increase the overall write endurance by 150× to a level that is sufficient for deployment. When compared to an earlier pure STT-RAM design [82] L1 cache proposal, our hybrid architecture achieved comparable performance and energy savings. However, we believe that because of the significant improvement in write endurance, our proposal is more practical, at least in the near term.

As the CMOS technology continues to scale, increasing energy consumption and design complexity demands for more power-efficient designs. We believe that our hybrid cache scheme is an attractive architecture for next-generation non-volatile computing.

Chapter 5

Architecting Multi-Level Cell STT-RAM as Last-Level Cache

In this chapter, we continue to move down to the next level in the memory hierarchy: *last-level cache*. At this level, data access is much less frequent and capacity becomes the main design target. Therefore, we shall choose a technology with the focus on high storage density. Multi-level cell (MLC) STT-RAM is such a choice that can achieve 2× the capacity as conventional *single-level cell* (SLC) STT-RAM under a similar silicon area budget. Figure 5.1 shows the scope of this chapter and the memory technologies involved.

However, most existing works concentrated on SLC design while the potential of MLC STT-RAM has not yet been fully explored. The main reason is the two-step RW operation which inevitably introduces performance and energy overhead, jeopardizing the gains from a higher capacity. In this work, we proposed an architectural level design to dynamically reconfigure the cache block size for a MLC-based STT-RAM last-level cache. Our approach places certain hot data

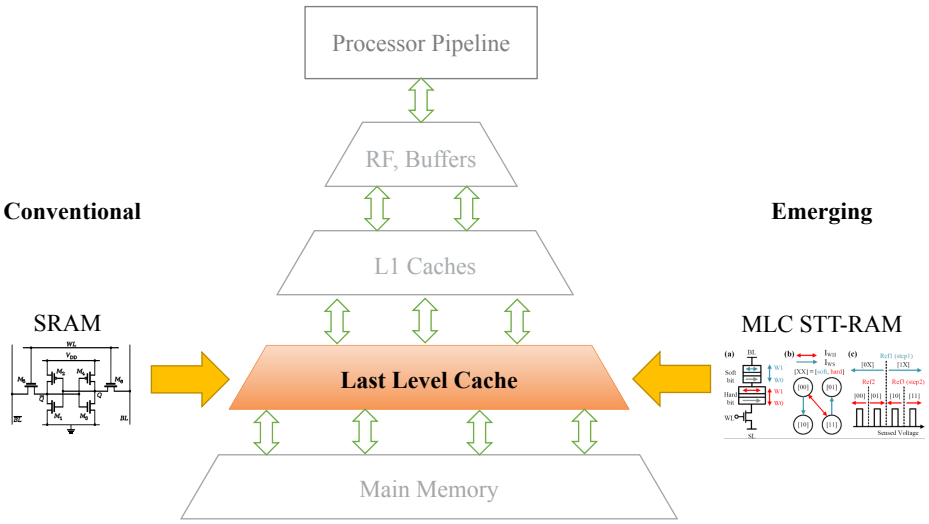


Fig. 5.1 Scope and memory technologies used in this chapter.

chunks in smaller blocks so as to benefit from the lower latency and energy, while keeping the rest in larger blocks to maintain an overall hit rate. As such, we mitigated the penalties of MLC STT-RAM while increased overall performance with less energy consumption.

5.1 High Capacity LLC using MLC STT-RAM

5.1.1 Block-level Data Mapping

As mentioned in section 2.2.4, each MLC STT-RAM can store 2 bits of data instead of 1 bit in SLC. Thus, while using MLC STT-RAM for cache, one of the first design decision is how to organize the layout of the data bits within a cache block. For example, given a 512-bit (64-byte) data block and 256 2-bit MLCs, one can store the i-th bit in the $i/2$-th MLC, as shown in Figure 5.2a. We shall refer to this organization as *direct mapping* (DM). DM is a straightforward

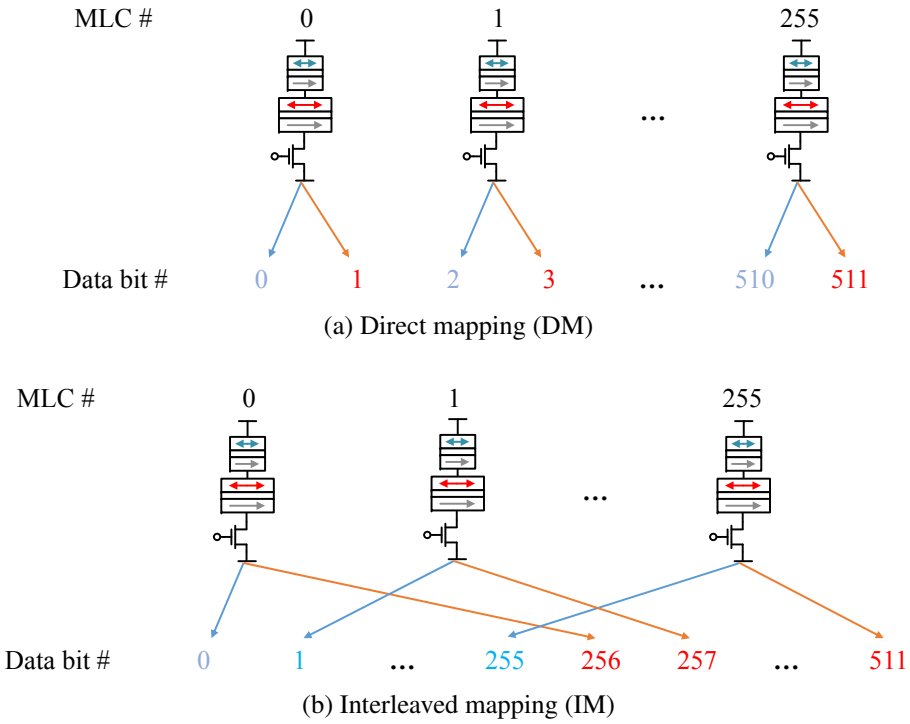


Fig. 5.2 Block-level data mappings for a 512-bit (64-byte) MLC cache block.

method without differentiating the accesses to different parts of the block. Therefore, the latency is always the worst-case since both the soft- and hard-bits need to be sensed/programmed together regardless which data word is requested.

As an alternative, we can put the lower half (bit 0-255) of the data bits in the soft-region of the MLCs and the higher half (bit 256-511) in the hard-region. This method is illustrated in Figure 5.2b as shall be referred to as *interleaved mapping* (IM). IM reduces the write latency and energy for the lower half of the block since only one-step programming is necessary for the soft-bits without alerting the hard-bits. However, read latency remains the same for both halves. Furthermore, there is no guarantee that the lower half of the block will endure

more writes than the higher half. Thus, the benefit of using such an organization alone is limited.

In addition, one can split the contents in a block and makes each MLC to store data across different blocks [37, 9]. However, such kinds of organization break the physical integrity of a data block and increase the complexity of wirings. Under the circumstance when a single MLC fails, two data blocks are affected. Throughout this study, we will maintain the physical integrity of data blocks and map each MLC to only one target due to reliability concerns.

5.1.2 Access Behavior in Last-level Cache

In modern multi-core processors, the access behavior of *last-level* cache is usually much more irregular than the upper level caches. For set-associate cache, the reads and writes are generally distributed non-uniformly across different blocks and sets in the workloads we have studied including solving PDE, fluid dynamics simulation, real-time raytracing and video encoding. Figure 5.3 illustrates four different access patterns found in the PARSEC benchmark suite. We recorded the number of accesses to each half of the block within a cache set. In (a), the lower half of block 0 received substantially more accesses compared to other blocks in the same set. In (b), the lower half of each block has $2\times$ activities compared to the higher half. In (c), most of the access hit on the first few blocks of the set, resulted in the ways being underutilized. The last case is a uniform access pattern where the accesses are spread out across the whole cache set.

Recall that in the use of MLC STT-RAM, we can either utilize the full 2-bit cell or simply keeping the hard-bit in a fixed resistance and use soft-bit only for

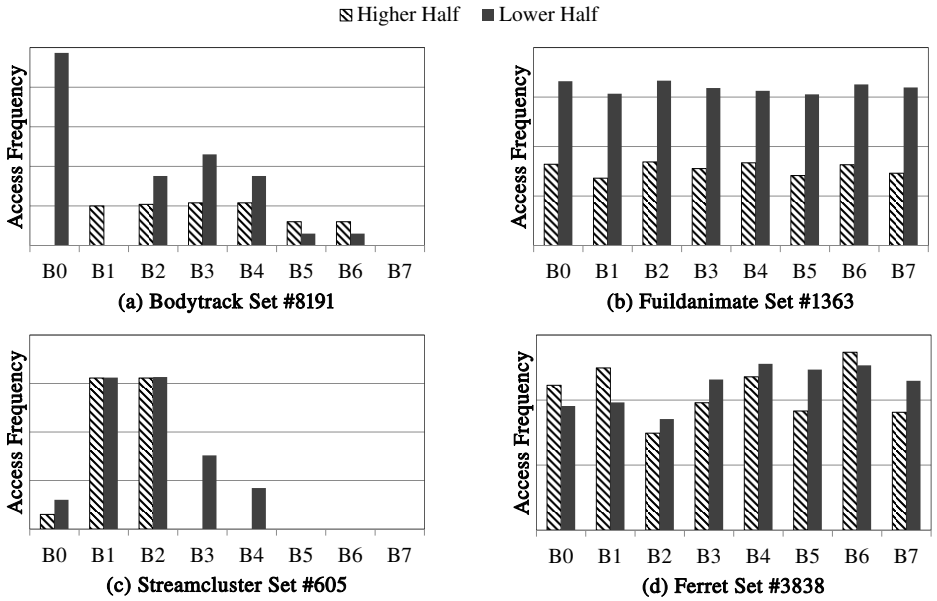


Fig. 5.3 Various access patterns observed in four benchmarks from PARSEC under an 8-way, 64-byte block, 4MB last-level cache.

accelerated access. In the above examples, (a) and (c) fail to utilize the whole set fully. Thus, a better performance could be obtained by reducing the capacity. In (b), it depends on the difference between the performance gain from a smaller set and the penalty caused by a higher miss rate. For (d), it is most likely that performance will be maximized if the set stays in a larger capacity.

Not only does the cache access pattern behave differently from set to set, but also the behavior of a cache set might change during different program phases. Figure 5.4 shows the access pattern changes in set #129 of *ferret* during the first 400 million CPU cycles. In the first and second phases (0-200M cycles), although block $0, 2$ and 3 together are much more active than the others, the overall access pattern tends to spread out and a larger capacity should be maintained. In the third phase (200-300M cycles), the accesses are concentrated in only three half blocks so thus the capacity is underutilized. Finally, in the last phase (300-400M cycles),

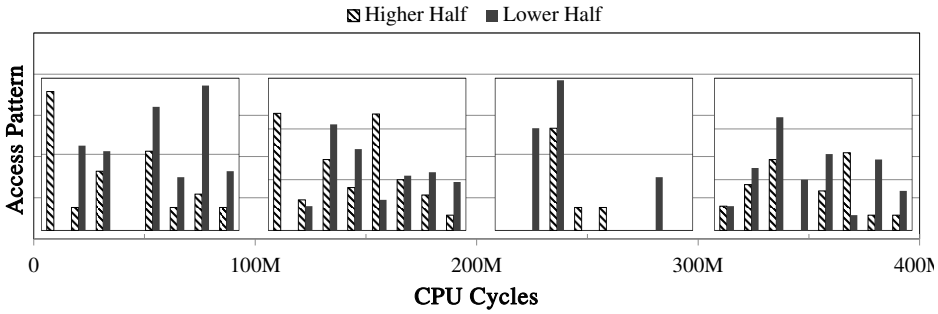


Fig. 5.4 Access pattern changes in set #129 of *ferret* for the first 400M cycles.

the accesses are spread out again. Hence, when facing such non-uniformities, system performance could be optimized if each cache set adapts its configuration dynamically responded to the access pattern.

5.2 Designing MLC-based STT-RAM Cache

In order to trade capacity for speed, previous work suggested splitting the block-level data mapping for the MLCs and turning off half of the ways to enable faster access [9]. However, this method reduced associativity and resulted in a waste of half of the tag array. Considering the accesses to a large data block are often distributed unevenly between the two halves, as shown in last section, it would be more resource-efficient to dynamically reconfigure the block size while keeping the same associativity.

5.2.1 Address Decomposition

With the help of interleaved mapping (Figure 5.2b), each cache set can operate in exactly one of the two modes: a *large block mode* (LBM) and a *small block mode* (SBM). LBM utilizes all the physical data bits (bit 0-511) in a MLC STT-

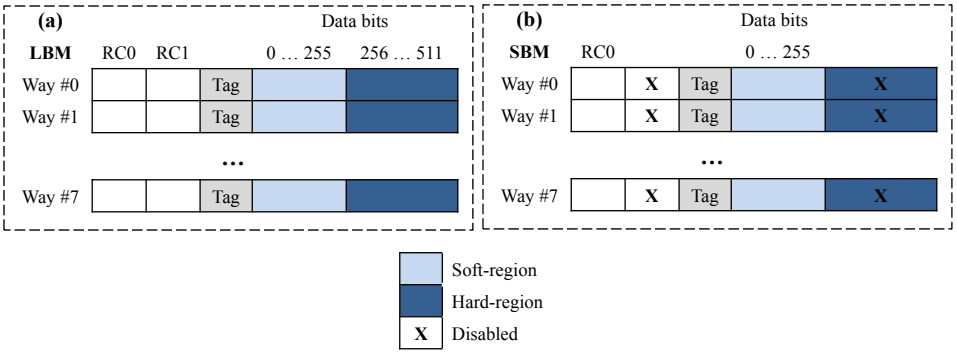
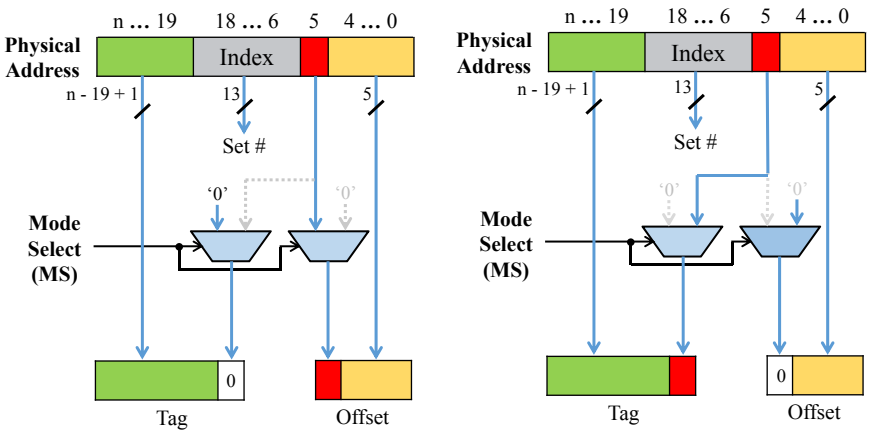


Fig. 5.5 Two operating modes for an 8-way cache set with different block sizes: (a) Large block mode (LBM); (b) Small block mode (SBM).

RAM data block and suffers the maximum access latency. SBM uses only the soft-bits (bit 0-255) and so is much faster and consumes less energy. Figure 5.5 demonstrates these two operating modes for an 8-way associative cache.

While both LBM and SBM can co-exist in a single cache, it is necessary to guarantee that no data chunk is mapped to more than one set. Figure 5.6 describes such a decomposition scheme supporting mixed block modes access given a n-bit physical address. Assuming the cache is composed of $8k$ sets, the set index is fixed at address bit 6-18 so that each memory location maps to only one cache set regardless of the block size. An additional control bit called *mode selection* (MS-bit) is used to indicate the block mode. When $MS = 0$, the set operates in LBM (Figure 5.6a) with a large 64-byte block size. A zero-bit is appended at the end of the tag and the offset is set to the lowest 6 bits of the physical address in order to select any byte within the block.

When in SBM ($MS = 1$) (Figure 5.6b) all blocks are halved at 32-byte. Additional information (bit 5 of the physical address) is added to the end of the tag to differentiate the lower and higher halves of a large block. Note that only a



(a) Large block mode (b) Small block mode

Fig. 5.6 Physical address decomposition designed for two set operating modes.

5-bit offset is now sufficient, so the highest bit is zeroed out. Figure 5.7 illustrates an example of address decomposition for a 32-bit physical address `0xc7a97eeb` which is mapped to the same set `0x5fb` with different tag and offset values under the two block modes.

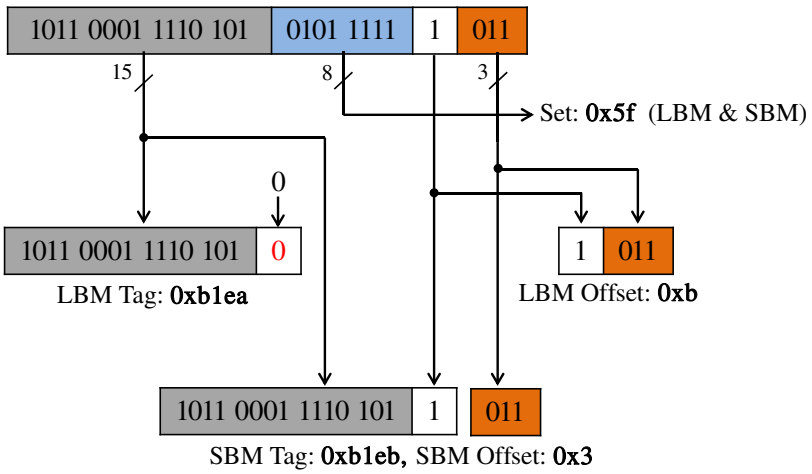


Fig. 5.7 Decomposition of address `0xc7a97eeb`.

5.2.2 Reconfiguration Strategy

Ideally, to determine which mode a cache set should stay during a period of time for maximum performance, we calculate the advantage of LBM versus SBM characterized as:

$$\delta_{LS} = (\#Extra\text{-}Misses\text{-}SBM) \times (Miss\text{-}Penalty_{avg}) - (\#Hits\text{-}SBM) \times (Latency\text{-}Reduction_{avg}) \quad (5.1)$$

Here *Extra-Misses-SBM* is the number of cache misses caused by reducing the block size. If δ_{LS} is larger than zero, staying in LBM is more beneficial, otherwise switching to SBM should yield a better performance. A prediction algorithm for δ_{LS} is as follows. Each cache block is equipped with two 2-bit saturating *reference counters*, namely RC0 and RC1 as shown in Figure 5.5. Under LBM, RC1 is responsible for the higher half of the block and RC0 is for the lower half while only RC0 will be used in SBM. When a block is initially loaded, its RCs are set to the maximum value (3). When an memory access hits on a block, it decreases the RC. Upon the RC saturates at zero, additional accesses to the same block shall increase all the other RCs within the same set, promoting itself to be a hot (half) block.

When a cache miss occurs, all of the RCs within the set are examined. For a cache set to be switched from LBM to SBM, the number of zeros among all the counters must be less than a predefined threshold θ_{LS}. For switching from SBM

to LBM, the number of zeros shall be larger than another threshold θ_{SL}. This process is shown in algorithm 2.

```
if Cache hit then
    if RC > 0 then
        Decrease it by 1;
    else
        Increase other RCs by 1;
    end
else
    if In LBM and 0 < #Zeros in RCs < θ_LS then
        Reconfigure to SBM;
    end
    if In SBM and #Zeros in RCs > θ_SL then
        Reconfigure to LBM;
    end
end
```

Algorithm 2: Block size reconfiguration policy.

A cache miss in a LBM set with very few zeros indicates the access to this set are concentrated only on certain hot (half) blocks. Therefore, reconfiguring the set to a smaller block size will not introduce significant misses and can possibly obtain performance/energy benefits. On the other hand, if there are too many zeros in a SBM set, each block could have been competing for accesses (Figure 5.3d). Thus, the miss penalty is predicted to be higher than the latency reduction and reconfiguring to LBM might be more profitable.

Housekeeping

During reconfiguration, dirty data blocks shall be first written back to memory, followed by a series of housekeeping operations. If the block size is reduced (LBM to SBM), a simple read-and-write scheme can be adopted to preserve the contents in the lower half block without changing the tag array. When going

the other way (SBM to LBM), re-fetching the other half for every data block is expensive. Since both the lower and higher halves of a large block may be in the set at the same time, it is necessary to spend extra efforts in tag checking to discover those relationships. Hence, we shall only re-fetch those blocks with 0 in the last bit of the tag and discard all the others. Furthermore, since a new data block will be brought in and might result in eviction of an old block, simply by discarding all the data is another choice to reduce the miss penalty incurred during reconfiguration.

Reconfiguration Timing

Reconfiguring the block size during cache misses results in a lower overhead. For LLC, a miss must be serviced via the main memory. Hence the existing penalty is already few orders of magnitude larger than checking the RCs. However, if the checking is performed during cache hits as well, this additional latency can no longer be ignored, and might have significant impact on performance. Furthermore, frequent checking might incur more noises in the prediction. It is therefore more profitable to perform the checking and reconfiguration only during cache miss servicing even though some beneficial reconfigurations might be delayed.

5.2.3 Other Design Consideration

In order to protect the cache set from harmful reconfigurations, an additional bit (PB) is added to each set. When a cache miss occurs, based on the current operating mode and the access counters, PB will be set to 1 if a reconfigurable

access pattern is detected. Only upon the next miss if the same pattern continue to exist will the set be reconfigured. Otherwise, PB is reset when the access pattern changed during the subsequent miss.

In general, the granularity of mode prediction is controlled by the width of the RC. For a m-bit RC, a (half) block requires at least $2^m - 1$ hits to be promoted as a hot (half) block. A larger value of m increase the confidence of prediction since it keeps track of more access information. However, it also increases access latency and delays the timing for the beneficial reconfigurations which make the design less effective. A 2-bit RC width is proved to be well effective in the experiments with very little performance impact.

5.3 Evaluation

5.3.1 Experiment Setup

We used the cycle-accurate simulator MARSSx86 [64] to evaluate our proposal. We modeled a conventional quad-core x86-64 processor with a two-level cache hierarchy. Nine diverse multi-threaded workloads from the PARSEC multi-threaded benchmark suite [10] were chosen for the experiment. System performance was measured based on *instruction per cycle* (IPC). The list of simulator parameters is given in Table 5.1.

Three MLC STT-RAM based cache designs are compared:

- **LBM Only**: Baseline MLC STT-RAM cache, the full 2-bit cell is utilized to enable the maximum capacity.

Simulation Parameters	
Processor	2GHz, 4 out-of-order cores
Pipeline	4-way issue, 14 stages
ROB / LSQ / In-flight Branches	128 / 80 / 24
Functional Units	2 ALUs, 1 FPU, 2 LUs, 1 SU
Branch Predictor	Gshare, 16 history bits, 4K BTB entries
I- / D-TLB	32 / 32
L1 I- / D-Cache (SRAM)	32KB / 32KB, 4-way, 32-byte block, 2-cycle latency
Coherent Protocol	Illinois MESI
Last-Level Cache (MLC STT-RAM)	Maximum 4MB, 8K sets, see Table II for the details
Main Memory	4GB, 290-cycle latency

Table 5.1 Key simulation settings.

- **Mixed Block Sizes (MBS)**: Our proposed design of mixed block sizes (LBM + SBM) with dynamic reconfiguration. We set both θ_{LS} and θ_{SL} to be half of the associativity. By default, all the cache sets operate in LBM.
- **ASM**: The alternative design proposed in [9], where some sets will be dynamically way-reduced to enhance accessing speed. We choose $M_{th} = 512$ as suggested in the paper.

The latency and energy numbers for MLC STT-RAM LLC are generated using NVSim [26] with the MTJ and CMOS technology parameters adopted from [9]. Table 5.2 shows these parameters for different set operating modes.

5.3.2 LLC Miss Rate and System Performance

Figure 5.8 shows the *misses per kilo-instruction* for the LLC of the baseline and our design. Six out of nine benchmarks did not incur noticeable increase of cache

LLC Parameters (8K sets, max 4MB capacity)			
Set Operating Mode	LBM	SBM	ASM
Set Configuration	8-way, 64-byte block	8-way, 32-byte block	4-way, 64-byte block
Read Latency (cycle)	10	7	
Write Latency (cycle)	44	23	24
Miss Latency (cycle)		4	
Read Energy (nJ)	0.543	0.419	0.424
Write Energy (nJ)	1.562	0.756	0.853
Miss Energy (nJ)	0.056		0.033

Table 5.2 LLC latency and energy numbers.

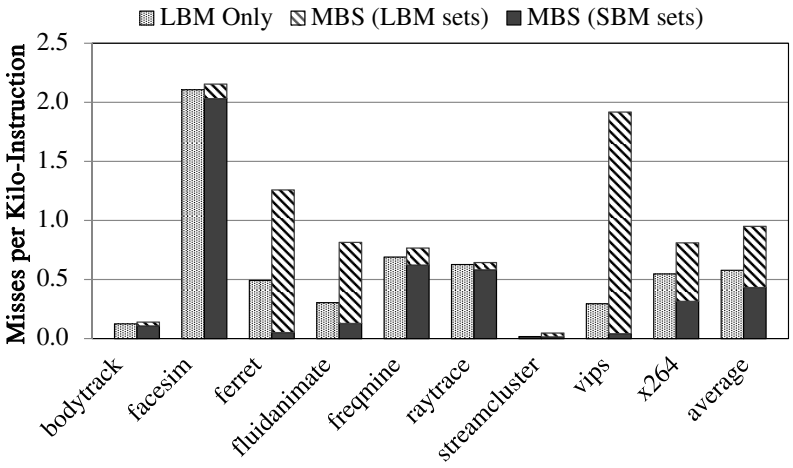


Fig. 5.8 LLC miss rate comparison.

misses. In particular, *bodytrack* and *streamcluster* has a very low miss rate since their working set did not exploit the full associativity. In the case of *facesim*, *freqmine* and *raytrace*, the portion of cache sets that were reconfigured to SBM is less than 10% and therefore most misses come from the LBM sets. On the contrary, more than 90% of the sets have been reconfigured to SBM in *ferret* and *vips*, resulted in a significant amount of miss increases.

Figure 5.9 shows the normalized IPC for our design and the competitor (ASM). Our scheme increases the IPC of baseline by 1% to 10% across different

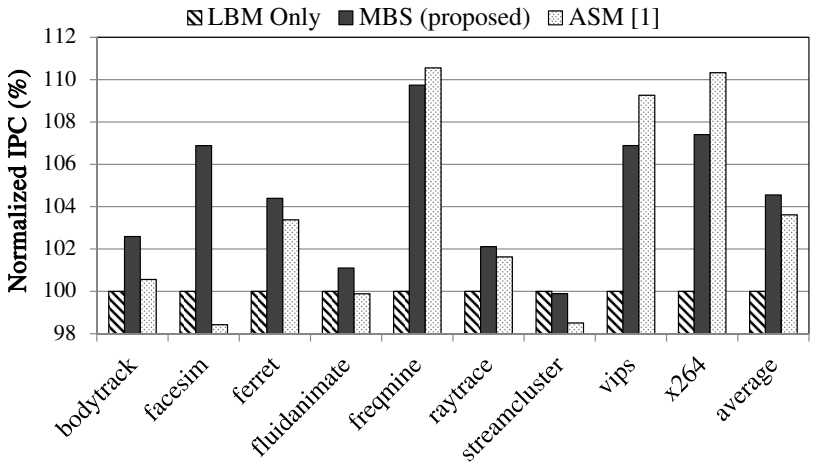


Fig. 5.9 Normalized IPC comparison.

benchmarks. In particular, *facesim* has the most number of sets with spread out access behavior and thus resulted in the least number of reconfigurations. However, these few number of sets that has been identified to benefit from SBM contributed a rather large increase in IPC (close to 7%). Furthermore, in the case of *ferret* and *vips*, despite that they suffered much larger miss rates, the IPCs are actually 4% and 7% higher. It demonstrates that not all cache accesses are equally performance critical. Our design correctly predicted the δ_{LS} value in determining a more beneficial block mode. On average, we increased baseline IPC by 4.6%, 27% more than ASM's improvement over the baseline.

5.3.3 Energy Consumption

Since STT-RAM is non-volatile, the main leakage energy comes from the peripheral circuits which is shared between the three designs. Thus only *dynamic energy* is discussed here. Figure 5.10 shows the normalized dynamic energy consumption including those spent in cache hit, miss handling and additional

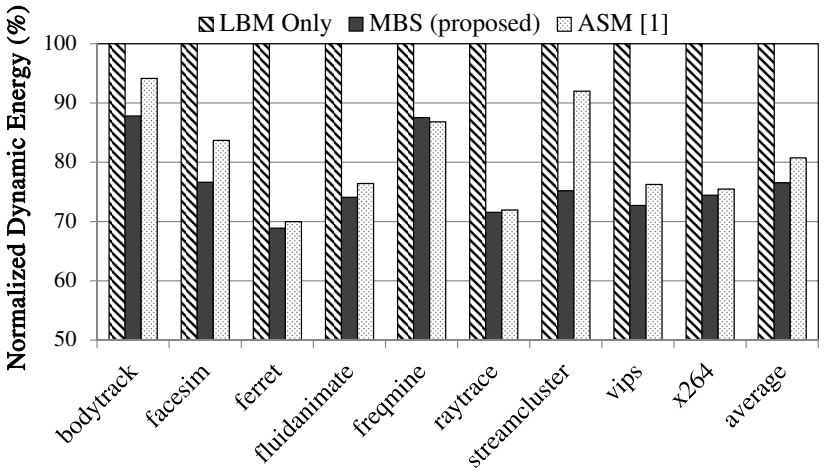


Fig. 5.10 Normalized dynamic energy comparison.

housekeeping operations during reconfigurations. The default LBM Only design consumes the most energy because of the extra dynamic energy spent in the two-step access is more than the additional miss energy incurs from a lower hit rate (MBS and ASM). The most reduction occurred in *ferret*, where more than 30% savings was recorded. The energy saving for our design is 23.5% on average while ASM consumes 5% more than ours.

5.3.4 Sensitivity of Reconfigure Threshold

Table. 5.3 shows the normalized IPC under different θ_{LS} and θ_{SL} values. Since our simulation setting uses an 8-way set-associative LLC, the reasonable maximum values of θ_{LS} and θ_{SL} were set to 8 and 4, respectively. Overall, performance increases as θ_{SL} becomes higher under all θ_{LS} values. This is because most of the cache sets exhibit the same consistent access behavior throughout the application runtime. In particular, when a set is firstly predicted to benefit more from reconfiguration, it is most likely the best choice for its subsequent life. As

θ_{LS}\ θ_{SL}	1	2	4
2	97.7%	98.2%	98.35%
4	97.4%	98.65%	100%
6	98.29%	98.4%	98.5%
8	97.8%	98.23%	99.1%

Table 5.3 IPC for different reconfigure threshold settings, normalized to θ_{LS} = θ_{SL} = 4

such, we can reduce the chance of reconfiguring the set back to LBM by setting a higher value of θ_{SL}. Based on the experiments, the optimal values of θ_{LS} and θ_{SL} are found at 4 for both.

5.4 Related Work

Device-level optimization for MLC STT-RAM has been proposed before [17, 98]. Architecture study of MLC STT-RAM cache were also demonstrated. Jiang et al. [37] categorized the MLC cache lines as *read-fast and write-slow* or *read-slow and write-fast* based on the splitting of hard-bits and soft-bits. Depending on the read/write activities of each cache block, data migration is triggered to optimize the overall performance. Chen et al. [18] has tackled the write endurance issue of MLC STT-RAM. Wen et al. [91] has developed a holistic solution set to improve data integrity and performance of MLC STT-RAM. Bi et al. [9] has proposed cross-layer solution including MTJ designs as well as architecture schemes for MLC STT-RAM caches.

5.5 Summary

The skewness of cache accesses exists in most if not all applications. In this work, we exploited such a characteristic in designing energy-efficient caches for emerging memory technology. In particular, we proposed an architecture-level design for MLC-based STT-RAM on-chip caches. Our scheme dynamically predicts the access behavior of cache sets and reconfigures them to appropriate block sizes for better performance and energy conservation. Simulation showed that compared to a conventional design, our method increased IPC by 4.6% with 23.5% reduction in dynamic energy consumption. As the CMOS technology continues to scale, energy and design complexity issues call for more power-efficient designs. We believe that such a reconfigurable cache built with the emerging STT-RAM technology is an attractive design option for next-generation computing.

Chapter 6

Architecting STT-RAM as Last-level Cache for GPGPU

In recent years, incorporating *graphics processor unit* (GPU) to support scientific computation has become the mainstream choice for many high-performance computing systems. *General-purpose GPUs* (GPGPUs) can now handle a wide range of non-graphic computations that were CPU-only in the past. They can easily outperform general CPUs on workloads that contain massive amounts of data-parallelism, for example, monte carlo simulation, weather forcasting, molecular modeling, quantum mechanical physics and computational finance. To support such a high amount of parallelism, the memory system of GPGPU requires a different design methodology from CPUs'.

In this chapter, we will focus on the last-level caches of GPGPUs. Having investigated the memory access behaviors of some GPGPU applications, we proposed an energy-efficient hybrid last-level cache design with conventional SRAM and emerging STT-RAM technology. Our scheme assumes each L2

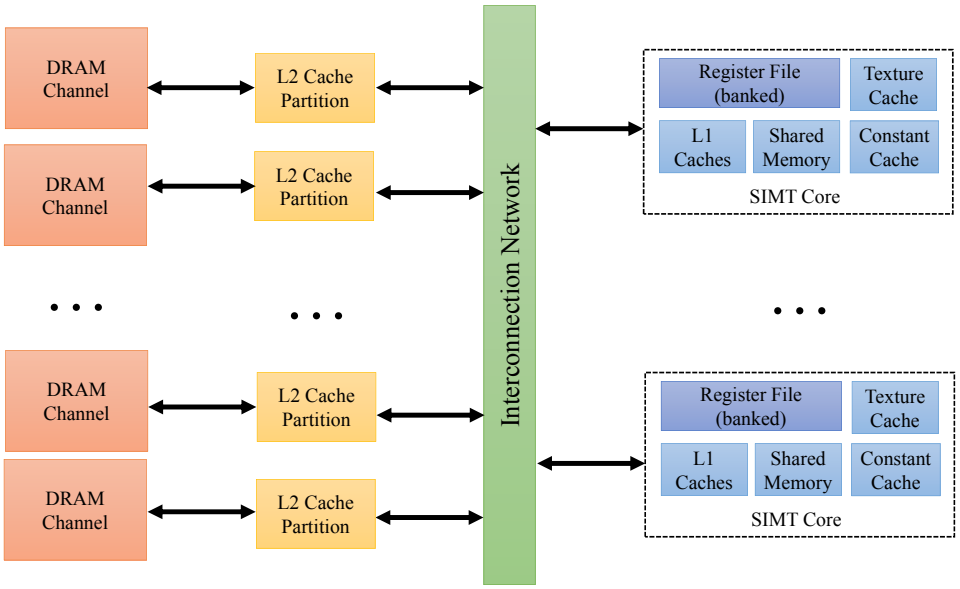


Fig. 6.1 GPGPU Memory Hierarchy.

partition consists a large STT-RAM part and a small SRAM augment. It includes three core techniques: *differential block allocation*, *block migration* and *pre-migration*. Compared to pure L2 cache based entirely on either SRAM or STT-RAM, our scheme achieved a better energy efficiency.

6.1 GPGPU Memory Hierarchy

The memory hierarchy of modern GPGPUs share some similarities to CPUs but is different in a number of aspects. A GPGPU is typically consisted of a number of *single-instruction multiple-thread* (SIMT) cores. Each SIMT core has its own register files, a scratch-pad and several first-level caches. These cores use a shared interconnection network for the communication to last-level cache and main memory. Since GPGPUs are designed for executing hundreds or even thousands of threads to achieve a massive amount of *thread-level parallelism*

(TLP), memory accesses are usually coalesced to exploit spatial locality and maximize peak bandwidth. Figure 6.1 shows a typical GPGPU memory hierarchy that consists of the following segments

- **Register File**: Unlike CPU where the register file is typically multi-ported, GPU's register file is multi-banked to enable simultaneous access for a collection of threads (warps).
- Scratchpad: This is also called *shared memory*. It is a software-managed memory that shared within a thread block. Note that scratchpad is not backed by L2 cache.
- L1 Caches: First-level caches including instruction cache and data cache.
- Texture Cache: Specialized cache for storing texture data.
- Constant Cache: Specialized cache for storing read-only data.
- L2 Cache: Last-level cache is divided into a number of partitions. The memory requests from previous-level are routed to them through the interconnection network.
- Off-chip Memory: Each L2 cache partition is backed by an individual DRAM memory channel to achieve high-bandwidth data transfer.

6.1.1 L2 Cache

With process scaling, GPU manufacturers are constantly increasing the number of execution units in the GPU core. For example, the 65*nm* NVIDIA Tesla-based GPU contains merely 128 shader units [53] while the latest 28*nm* Maxwell-based GPU has exceeded 3000 shader units within a single processor die. At the same time, the increasing amount of TLP puts more pressure on the memory system. In particular, as the capacity of L1 cache is limited and cannot fit in a large

Workload	L2 Cache			
	1MB	2MB	4MB	8MB
bfs	1.0	1.43	1.52	1.56
b+tree	1.0	1.03	1.06	1.09
cfd	1.0	1.03	1.13	1.26
dwt2d	1.0	1.13	1.3	1.4
hybridsort	1.0	1.17	1.5	1.5
lud	1.0	1.03	1.03	1.03
pathfinder	1.0	1.05	1.07	1.09
srad	1.0	1.03	1.1	1.12
histo	1.0	1.06	1.52	1.6
stencil	1.0	1.31	1.66	1.66
sad	1.0	1.2	1.39	1.6

(a) Normalized performance

Workload	L2 Cache			
	1MB	2MB	4MB	8MB
bfs	1.0	0.92	1.15	1.87
b+tree	1.0	1.21	1.68	2.95
cfd	1.0	1.18	1.49	2.34
dwt2d	1.0	1.14	1.45	2.54
hybridsort	1.0	1.09	1.26	2.27
lud	1.0	1.23	1.78	3.32
pathfinder	1.0	1.19	1.64	2.92
srad	1.0	1.23	1.67	3.02
histo	1.0	1.16	1.18	1.94
stencil	1.0	0.97	1.03	1.81
sad	1.0	1.08	1.36	2.2

(b) Normalized enrgy

Fig. 6.2 Normalized performance/energy for various L2 cache sizes.

working set, increasing the size of L2 cache becomes the preferred choice. As an example, NVIDIA doubled the amount of L2 cache from 768KB in Fermi-based GPUs to 1.5MB in Kepler-based GPUs [92]. Figure 6.2a shows the normalized performance for various L2 cache sizes tested on our baseline GPGPU system (Section 6.4.1).

Due to the high amounts of memory contention in shared caches in GPGPUs [19], enlarging the shared L2 cache can significantly speedup a wide range of applications including searching, sorting, data mining, fluid dynamics, bioinfomatics and image processing [16, 79]. Our experiments shows that performance is increased by 3% - 66% for different workloads when the L2 cache enlarged from 1MB to 8MB. However, notice that such performance gains comes with the expense of energy. For SRAM-based cache, a larger capacity means a larger

leakage power which is starting to dominate overall every consumption. For example, Figure 6.2b shows that 3× more energy is consumed in exchange of 12% performance in *srad*. Interestingly, *bfs* and *stencil* has less energy consumption in 2MB than 1MB. This is because the extra dynamic energy consumed in a 2MB cache is less than the static energy saved from a faster time of completion. Such a *sweet spot* is not always there in every workload. Overall, as L2 cache has already consumes 20%-40% of the total energy within the GPU chip [35], blindly increasing its capacity is not an wise way to improve the overall system performance.

6.1.2 Emerging Memories in GPGPUs

Incorporating emerging memories such as STT-RAM into the design of GPGPU memory hierarchy has been studied before with the focus on register files [50, 33, 42]. While the capacity of L2 cache is crucial to performance, our work shall spend efforts in improving the energy efficiency of L2 with the help of emerging memories.

6.2 Memory Access Patterns of GPU Applications

As mentioned in previous chapters, STT-RAM has several advantages such as low leakage power and smaller feature size over SRAM when used in constructing energy-efficient cache systems. Again, the key is to mitigate the infamous write penalty. Having examined the memory access traces of some GPGPU applications running on the simulation platform, the following access patterns have been observed at the block-level:

Read-intensive: The read-intensive pattern represents the case that during a cache block's lifetime, it is read much more often than is written to or even without writes (instruction/constant data). This is illustrated in Figure 6.3. The x-axis is the time elapsed in clock cycles. The first row shows a read-intensive behavior where the block has been read 8 times with only 1 write from the start to cycle 7000. Such a pattern is ubiquitous in almost every GPGPU applications and occurs at various program stages. Since STT-RAM is more efficient in processing memory read, we should allocate/migrate such type of data block to STT-RAM.

Burst writes: The burst write pattern means there is a frequent access of writes during a short period of time. This is observed in cycle 1800-2800, 3900-4000 and 6000-7000 in the second row of Figure 6.3. Such a pattern occurs while the application is transiting from one computation stage to another, saving intermediate results. We should avoid allocating such blocks in STT-RAM.

Periodical read-write: This access pattern is demonstrated in the third row of Figure 6.3. The access trace can be divided into a number of groups and each group contains the same sequence of *reads followed by writes* operations. In the figure, the same read-write sequence occurs three times and each sequence contains three consecutive reads followed by one write. Such blocks can be allocated to STT-RAM only if the energy penalty of write is smaller enough than the energy savings from read operation.

Transit: In application like *cfd* and *pathfinder*, a large portion of the working set is transit data. These data does not get much reuse after loaded into the cache, hence they will not benefit from a larger capacity. Obviously, STT-RAM

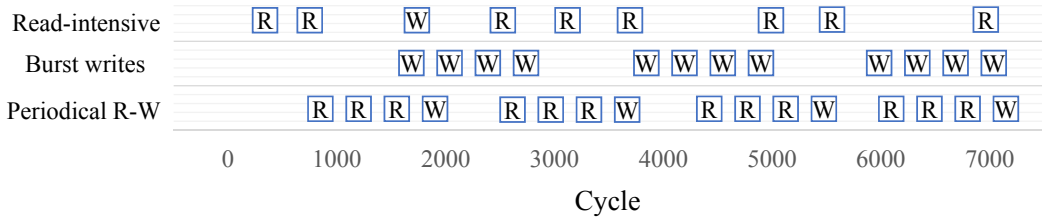


Fig. 6.3 Memory access patterns observed in L2 cache.

is not suitable to handle such kind of transit traffic because each cache line fill is considered as a write operation.

Irregular: All the other access patterns that do not belong to the above categories.

6.3 Proposed Architecture

In this section, we present a last-level cache architecture for GPGPUs. Our design employs a large-capacity STT-RAM sub-partition with a small SRAM augment in each of the L2 partition. The main design purpose is to keep those frequently used blocks with a high *read-to-write* ratio in STT-RAM so to benefit from the lower read energy and to reduce miss-rate, while allocate/migrate the rest that incur high write activities to SRAM.

Figure 6.4 depicts the internal structure of a L2 cache partition. It consists of a large STT-RAM sub-partition, a small SRAM augment and a allocation/migration controller (AMC). Both the STT-RAM and SRAM sub-partitions contain *tag*, *data* and *reference counter* (RC) arrays with different capacities. The RC array monitors the read and write activities occur at block-level. Our scheme employs two independent saturating counters in the RC array for read and write operation

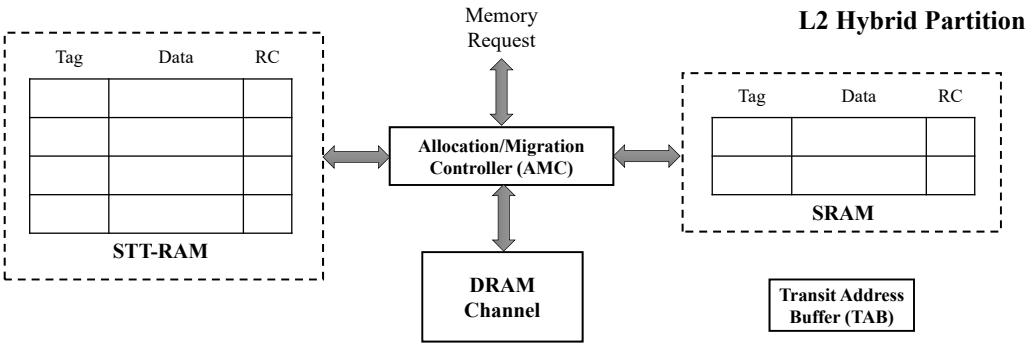


Fig. 6.4 The proposed L2 cache architecture.

respectively. Each cache access increases/reset the corresponding RC and based on such information, AMC migrates cache blocks when certain condition is met.

Differential Block Allocation. We observed that the miss type of a cache block can be used to predict its future behaviors. For write-miss, there are two categories of block encountered in L2: those that also miss first-level cache and those generated from L1's *write-evict* policy. In the former case, the write typically happens between the end of a thread block and the start of next thread block. Such a situation often indicates the written block is for storing the (intermediate) results from a series of computation and does not get reuse frequently. Therefore, by allocating them in the SRAM augment could avoid some non-profit cache fills (write) in STT-RAM.

In the latter case, the block has been evicted from L2 but still presents in L1, meaning the traffic (read) to it has been mainly absorbed by first-level cache and thus results in very few reuses from L2. Such a write-miss is typically followed by a read (if reuse) and/or an eviction (end of life) due to the conflict miss with other blocks. In both scenarios, the block is considered to be infrequent-read and shall be allocated in SRAM as well.

For read-miss, the block requested is assumed to be frequently-used with a high read-to-write ratio. They are allocated in STT-RAM at first but are subjected to migrations described in the next section.

Block Migration. It often happens that during a cache block's life time, its dynamic behavior changes from benefiting STT-RAM to penalizing STT-RAM. Thus a migration scheme is necessary to reduce such energy/performance penalties. Since most of the data blocks loaded to L2 cache are from read-miss, it is more important to identify those blocks in STT-TAM that incurs higher write penalty in the near future. This process is done by updating and examining the RC array. The read-RC and write-RC are initialized to zero when the block is firstly loaded. A read/write operation to the block will increase the corresponding RC by one. For blocks in STT-RAM, upon write-RC is saturated at a pre-defined maximum value θ_{rw}, the block will be migrated to SRAM with both RCs reset to zero. If read-RC is saturated first, both RCs are reset. On the SRAM side, the roles of read-RC and write-RC are revered as the migration to STT-RAM is triggered by a saturated read-RC value θ_{wr}.

Figure 6.5 demonstrated how a sequence of operations updates the RCs and triggers a migration, given a 2-bit wide read-RC and a single-bit wide write-RC. Here we set $\theta_{rw} = 1$. The block is initially allocated in STT-RAM part and behaves as read-intensive until event #7. Then it encounters a behavior changes and is migrated out. Figure 6.6 illustrates the data flows involved during a migration from STT-RAM to SRAM. Note that as now we have two parallel structures, probing has to be performed in both STT-RAM and SRAM part with different set index and address tag.

Event Sequence	Read-RC	Write-RC
1: Read miss	00	0
2: Read hit	01	0
3: Read hit	10	0
4: Write hit	11	1
5: Read hit	Read-RC saturated, reset	
6: Read hit	01	0
7: Read hit	10	0
8: Write hit	10	1
9: Write hit	Write-RC saturated, migrate to SRAM	

Fig. 6.5 An example event sequence that triggers a migration.

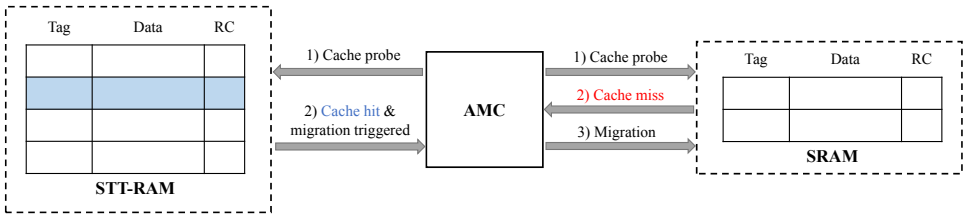


Fig. 6.6 Data flows involved when a migration is triggered from STT-RAM to SRAM.

The above scheme monitors the read and write activities occurs in a small window and uses it to predict the access behavior in a near future. If a cache block shows a heavy write activity, it is likely the block will hamper energy efficiency from staying in STT-RAM and vice-versa. However, migration is not free and incurs performance and energy overhead. If the block does not behave as write-intensive after being reallocated to SRAM, migration traffic is wasted and it might obstruct further memory requests to SRAM. In the experiment, we found that the number of blocks that require such kind of write-induced migration is generally small compared to the number of memory accesses. The additional

latencies caused by block migrations did not lead to notifiable performance degradation as we shall show in the evaluation.

Transit address buffer. Until now we are able to identify most long-term memory access behaviors and act accordingly, however, transit traffic can still cause unnecessary cache fills in STT-RAM and severely affects energy consumption. Here we introduce *transist address buffer* (TAB), a fully associative buffer for holding transit data address. When a block gets evicted with zero read- and write-RC from the STT-RAM part, the corresponding set index will be put into TAB. Any block fill to STT-RAM will first consulate TAB and if the set index matches an entry in TAB, that block will be allocated to SRAM augment with the TAB entry be removed.

Pre-Migration. For most GPGPU applications, the memory accesses to L2 are designed to be uniformly distributed to each partition (roughly) and these partitions share similar access behaviors. Thus, if a block migration is triggered in one partition, the corresponding blocks in other partitions are also likely to be migrated through subsequent memory accesses. In such cases, it would save more energy if the subsequent migrations in other partitions can be performed in advance. To implement this, migration signals are issued to all other partitions with the appropriate triggering memory addresses. Note that since all L2 partitions adopt the same configuration, only set index and address tag are needed.

6.4 Evaluation

In this section, we evaluate our proposed L2 architecture using a modified version of the cycle-accurate GPGPU simulator *GPGPU-Sim* [8]. It provides a detailed architecture model (including the internals of SMs, caches, interconnection networks and off-chip DRAM) of modern GPGPU running CUDA workloads [92]. We have chosen two widely used CUDA benchmark suites: *rodinia* [15] and *parboil* [79] for the experiments. The benchmarks are compiled using CUDA toolkit 4.0 [2] in order to satisfy GPGPU-Sim's requirement. The latency and energy numbers of SRAM and STT-RAM cache are generated using *DESTINY*, a tool for modeling emerging memories [67].

6.4.1 Configuration

Figure 6.7a shows the base GPGPU system configuration which mimics a Feimi-based microarchitecture [92]. Although the feature size of a STT-RAM cell is about $1/4$ of SRAM [82], same-capacity comparison was performed for fairness reason. Note that even after adding the SRAM augment in hybrid cache, the total area is still smaller than a single 4M SRAM cache. Three L2 configurations were simulated in the experiment:

- **SRAM**: A 4-megabyte pure SRAM cache with the maximum area occupation.
- **STT-RAM**: A 4-megabyte pure STT-RAM cache with the minimum area occupation.
- **Hybrid**: A hybrid cache composed of a 4-megabyte STT-RAM part and a 512-kilobyte SRAM augment, using differential block allocation and migration scheme described in previous section.

GPGPU Base Configuration	
SIMT cores / frequency	8 / 1GHz
Registers	32,768, 16 banks
Warp size / max threads	32 / 1024
L1 cache / shared mem.	16KB, 4-way, 128-byte / 48KB, 32 banks
Constant / texture cache	8KB, 2-way, 64-byte / 16KB, 4-way, 128-byte
L2 partitions (DRAM channel)	8
ROP / DRAM latency	100 / 200

(a) System parameters;

L2 Cache Configuration			
	Baseline SRAM	Baseline STT-RAM	SRAM Augment
Capacity	4MB		512KB
Associativity	8-way, 128-byte		
Latency (cycle)	Read: 5 Write: 5	Read: 4 Write: 30	Read: 4 Write: 4
Dynamic Energy (nJ)	Read: 1.32 Write: 1.27	Read: 1.0 Write: 15.1	Read: 1.2 Write: 1.15
Leakage Power (W)	0.45	0.013	0.128

(b) L2 cache parameters.

Fig. 6.7 GPGPU configuration.

The detailed latency and energy numbers are shown in Figure 6.7b.

6.4.2 Performance

To evaluate the performance impact from the proposed L2 architecture, we measured the execution time of each benchmark in GPU cycles. Note that GPGPU-Sim excludes the time spent in CPU-GPU communication so it only counts the cycle when GPU is busy. Figure 2 shows the normalized execution time of each benchmark. In a pure STT-RAM setting, the prolonged write latency did not cause significant performance degradation. The maximum execution time increase is merely 2%, recorded in *b+tree*. This is because write operations usually are not on the critical path and the interconnection and queuing delay of L2 access are essentially much larger than the programming delay of STT-RAM cell. The main penalty comes from the prolonged miss handling time which increases the latency of subsequent memory request along the execution path.

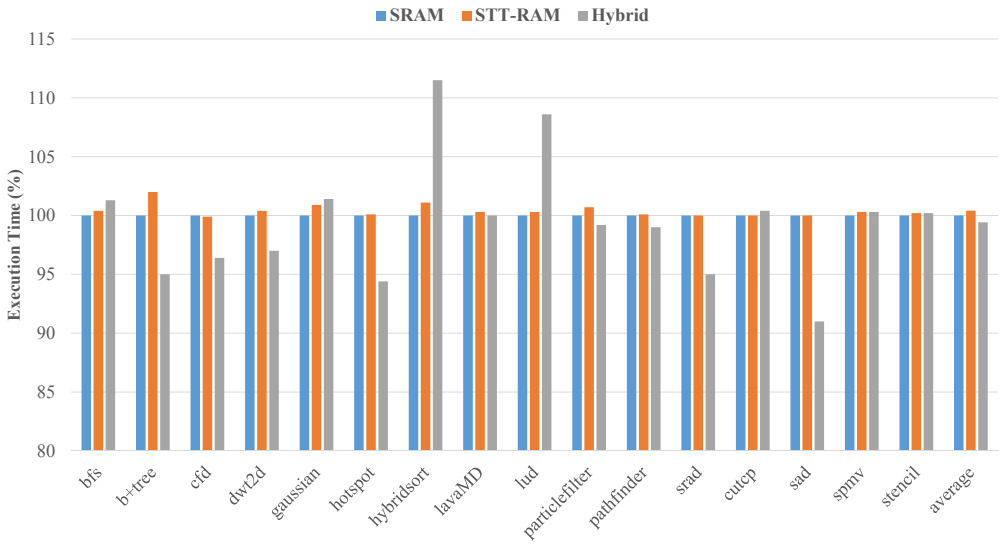


Fig. 6.8 Normalized GPU execution time.

As our design migrates writes to SRAM augment and reduces some traffic to STT-RAM part, the execution time of some benchmarks received minor deductions. In particular, *hotspot* and *sad* encountered a clean separation of reads and writes in STT-RAM part and SRAM augment respectively, shortening the total execution time by 5% and 8.8% respectively. The largest performance drop occurred in *hybridsort* and *lud*. Unlike other applications that seldom re-use those heavily-written data blocks, *hybridsort* and *lud* re-read great amount of such data. While they also exhibit a relatively high migration rate, the small SRAM augment is congested, causing a substantial increase of cache misses. As such, *hybridsort* and *lud* increase the execution time by 11.5% and 8.6% respectively. Overall, our design still saves slightly more than 1% of the total busy cycles.

6.4.3 Energy Consumption

The total energy consumption for the L2 cache is calculated by adding three components:

- **Leakage energy**: The static energy consumed when the data cell is not in operation. Note that due to the peripheral circuits, STT-RAM is not totally leakage-free.
- **Read/write energy**: Dynamic energy consumed for read and write operation. Note that every cache miss is considered to have an extra write due to block fill.
- **Probing and migration overhead**: In our proposed hybrid scheme, the AMC probes both STT-RAM part and SRAM augment in parallel when serving a memory request. The additional energy spent in the double-probings are considered here together with the energy spent in block migrations.

Figure 6.9 shows the normalized L2 energy consumption. As expected, the pure STT-RAM cache (red color) is less energy-efficient than pure SRAM in general because the high write energy consumed in STT-RAM is often much larger than the saving of leakage energy from SRAM. In benchmark *lud*, *particlefilter* and *cutcp* where the writes occurred less frequent (not necessary to be read-intensive), STT-RAM is more energy-efficient. Note that the write traffic is contributed by both memory writes and cache misses. In applications like *dwt2d*, *pathfinder* and *histo*, high portion of transit-like traffic were observed, leading to large amounts of block fills in STT-RAM and pushing up the total energy consumption. On average, a direct replacement with STT-RAM is not profitable as it 20% more energy.

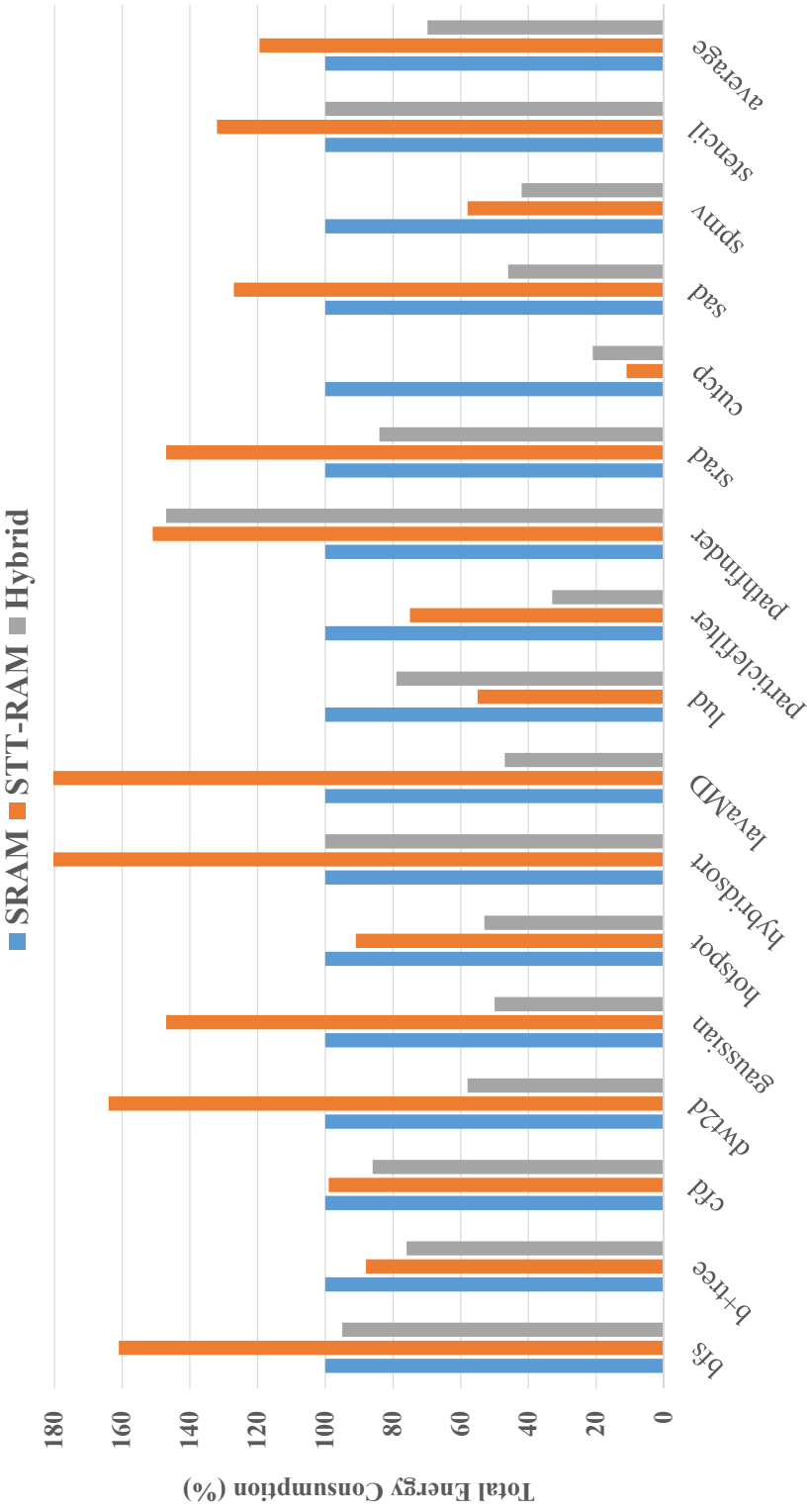
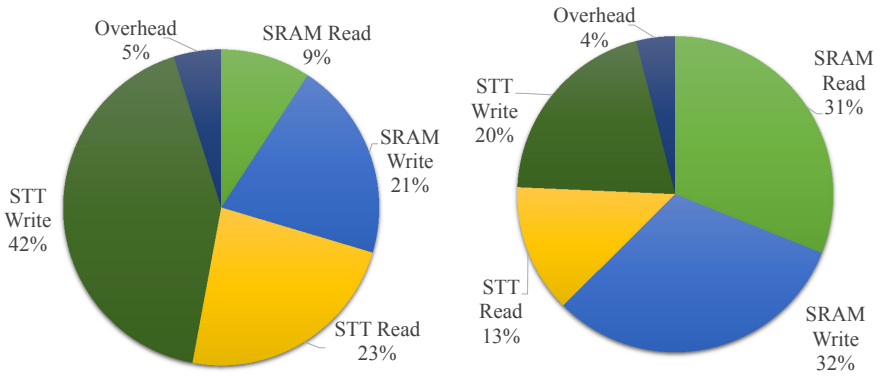


Fig. 6.9 Normalized total energy consumption of L2 cache.



(a) bfs (b) gaussian

Fig. 6.10 Dynamic energy distribution of hybrid L2.

Our design gradually reduce the write energy consumed in STT-RAM. The differential block allocation and migration scheme managed to offload portions of the write traffic to SRAM augment, saving the write energy of STT-RAM. This is shown in Figure 6.10 which displays the dynamic energy distribution of our hybrid L2 in *bfs* and *gaussian*. *Bfs* is write-intensive benchmark with medium portions of read/write mixed blocks. Our hybrid scheme allocated/migrated more than 80% of the writes to SRAM, while the rest still consumed more than 40% of the dynamic energy. *Gaussian* is also considered to be write-intensive, but 90% of the writes had been offloaded to SRAM augment. Thus, more dynamic energy was spent by SRAM augment. In both cases, the energy overhead incurred from double probing and migration is not significant, compared with the savings. In two special cases: *cfd* and *dwt2d*, the read and write data are separated in different memory region, making nearly all the write requests to be missed. Hence, STT-RAM part received almost zero write activity because of the differential block allocation.

6.5 Related Work

The use of emerging memories in GPGPU has been studied before. Li et al. proposed a hybrid STT-RAM and SRAM register file architecture leveraging the warp scheduler [50]. Their design utilized SRAM as a write buffer coupled with a warp-aware write-back strategy to reduce the high write energy consumed in STT-RAM. Goswami et al. implemented a STT-RAM-based register file with coalescing control strategy [33]. They attempted to remove redundancy updates to the register file by introducing a write-back buffer. Alternatively, Jing et al. presented another register file architecture design employing eDRAM [42]. They proposed bank bubble and walk-though technique to reduce the refresh overhead of eDRAM. Mao et al. demonstrated another power-efficient register file design using domain-wall memory [54] while Venkatesan et al. utilized DWM in the design of a hybrid cache hierarchy [85]. Furthermore, the attempt to use STT-RAM as GDDR5 replacement in graphic memory has also been investigated by Zhao et al. [99].

A last-level cache design with STT-RAM was given by Samavatian et al. [73]. In their work, both high-retention (HR) and low-retention (LR) STT-RAM cell were used. As described in chapter 2, relaxing the volatility of STT-RAM can significantly improve write performance and reduce write energy. Therefore, a monitoring logic was used to identify the write-intensive blocks and place them in the LR part. In addition, retention counters were introduced for postponing the refresh operation to the last cycles in the retention period, mitigating performance and energy overheads.

6.6 Summary

As GPU architectures scale to include more and more cores, enlarging on-chip memory to ensure sufficient data supply seems inevitable. While the emerging STT-RAM has shown promise as a replacement for SRAM in caches, at the current stage, an one-for-one replacement is still not profitable due to the cost of write operations in STT-RAM. A lot of efforts have been spent in mitigating this issue for CPU caches. However, due to the differences in architecture and application behaviors between CPU and GPU, these proposals do not translate automatically to GPGPUs.

In this chapter, we proposed a hybrid L2 architecture employing a large STT-RAM and a small SRAM section for GPGPUs. We showed that using our proposed differential block allocation and block migration scheme, we can alleviate the write pressure on the STT-RAM section significantly. Our design monitors the read/write behavior of cache blocks in STT-RAM, and attempts to offload writes to SRAM. Simulation shows our design achieved a small performance improvement over both pure SRAM and STT-RAM, but more importantly significantly reduces the energy consumption. We believe that this will contribute to the energy issue especially in integrating GPUs into systems-on-a-chip.

Chapter 7

Conclusion

7.1 Thesis Summary

Concerns about the energy issue are growing for modern processors at deep sub-micron era. Emerging memory technologies appear to be a promising way out. While they have many desirable properties, the drawbacks involved could hinder their future deployments. In this thesis, we have proposed four memory architecture designs utilizing emerging memories. The motivation is to maximize the energy and density benefits while mitigating the potential penalties. In particular, our contributions are

- At the highest level in the memory hierarchy, we have designed an analog neural branch predictor employing memristors. The characteristics of memristor allowed us to combine computation and storage all together. Our proposal utilized analog circuit technique in solving the latency issue found in most neural prediction algorithms. At the same time, its behavior is closer to neural systems in nature and thus resulted in a better prediction

accuracy. The non-volatile and low-power property of memristor also reduced the overall energy consumption by two orders of magnitude.

- At the next level in the memory hierarchy, we have designed a coherent hybrid L1 cache architecture employing both SRAM and STT-RAM. Our proposal made use of the built-in cache coherence protocol in predicting a cache block's dynamic behavior. To achieve a better energy-efficiency, writes are migrated to SRAM while most reads stay in STT-RAM. Two migration policies together with a null policy are presented and discussed. Experiment showed that when compared to conventional SRAM caches, our design saved significant energy while keeping the performance loss at a reasonable level.

- At the level of LLC, we have designed a MLC STT-RAM cache with dynamic block reconfiguration. LLC is typically large in capacity but is not often well-utilized. MLC STT-RAM can be configured into either a low-latency low-energy mode or a high-capacity mode. Based on that, we proposed a LLC design where some heavily used blocks are configured to full-capacity mode for a better hit rate while those under-utilized blocks are in low-latency mode for accelerated access. Simulation showed that such a design increased both system performance and overall energy-efficiency simultaneously.

- In addition, we have also extended the use of emerging memories to the memory hierarchy of GPGPUs. We proposed a high-capacity L2 cache for GPGPU using a large STT-RAM part coupled with a small SRAM augment. The main purpose is to alleviate the write pressure of STT-

RAM with the help of a differential block allocation and a block migration scheme. Simulation showed that our design achieved minor performance improvement over both pure SRAM and STT-RAM designs with significant energy reduction.

7.2 Future Direction

The proposed schemes are mainly focuses at higher levels in the memory hierarchy. Our main goal is to reduce energy consumption without compromising performance, while maintaining transparency to existing applications. As these emerging memories are not fully mature, *reliability* is an issue we have not investigated in this thesis. In addition, pure hardware mechanisms might not fully unleash the potential of emerging memories. A hardware-software co-design may further improve performance and energy-efficiency.

Reliability. The reliability of emerging technology such as STT-RAM is an important issue that constrains its commercial deployment. As the manufacturing process become more and more mature, *hard-errors* caused by limited endurance are expected to be lowered to a level that will work for most applications. Currently, the main source of errors are the *soft-errors* which are caused by *process variation* and *thermal fluctuation*. These errors can be categorized into three types: *read failure*, *write failure* and *retention failure*. Among them, retention failure has been shown to be the dominating source of error as technology scales [62]. At the moment, the lack of an accurate error model has limited further investigation to this issue. One possible solution is to employ multi-bit *error-correcting codes* (ECC) in the designs [90].

Hardware-software co-design. Being application-transparent, hardware mechanisms can be applied to any softwares without the source codes. However, as the hardware is not possible to be fully aware of the application behaviors, any decision made is not guaranteed to be always optimal. If the underlying hardware is exposed to upper-level (for example via the compiler) software, by providing application hints to the hardware we may reduce unnecessary (re)configuration overheads [52].

List of Publications

J. Wang, Y. Tim, W.-F. Wong, and H. H. Li, "A practical low-power memristor-based analog neural branch predictor," in *IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2013, pp. 175–180

J. Wang, Y. Tim, O. Zhong-Liang, W.-F. Wong, S. Zhenyu, and H. H. Li, "A coherent hybrid sram and stt-ram l1 cache architecture for shared memory multicores," in *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014, pp. 610–615

J. Wang, P. Roy, W.-F. Wong, X. Bi, and H. Li, "Optimizing mlc-based stt-ram caches by dynamic block size reconfiguration," in *32nd IEEE International Conference on Computer Design (ICCD)*, 2014, pp. 133–138

P. Roy, J. Wang, and W.-F. Wong, "Program analysis for approximation-aware compilation," in *ACM International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, 2015

References

[1] "Breakthrough nonvolatile memory technology," https://www.micron.com/about/innovations/3d-xpoint-technology.

[2] "Cuda toolkit 4.0," https://developer.nvidia.com/cuda-toolkit-40.

[3] "Intel's cpu roadmap: To nehalem and beyond," http://www.hardwarezone.com.sg/feature-intels-cpu-roadmap-nehalem-and-beyond/nehalems-core-and-tri-level-cache-structure.

[4] "Mram-info," http://www.mram-info.com/.

[5] "Amd64 architecture programmers manual volume 2: System programming," 2006.

[6] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock rate versus IPC: The end of the road for conventional microarchitectures," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, 2000, pp. 248–259.

[7] S. Ahn, Y. Song, C. Jeong, J. Shin, Y. Fai, Y. Hwang, S. Lee, K. Ryoo, S. Lee, J. Park *et al.*, "Highly manufacturable high density phase change

memory of 64mb and beyond," in *IEEE International Electron Devices Meeting, IEDM Technical Digest*, 2004, pp. 907–910.

[8] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2009, pp. 163–174.

[9] X. Bi, M. Mao, D. Wang, and H. Li, "Unleashing the potential of mlc stt-ram caches," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 429–436.

[10] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008, pp. 72–81.

[11] Z. Biolek, D. Biolek, and V. Biolková, "Spice model of memristor with nonlinear dopant drift." *Radioengineering*, vol. 18, no. 2, 2009.

[12] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 52.

[13] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 11, no. 3, p. 28, 2014.

[14] M.-T. Chang, P. Rosenfeld, S.-L. Lu, and B. Jacob, "Technology comparison for large last-level caches (l 3 cs): Low-leakage sram, low write-energy stt-ram, and refresh-optimized edram," in *Proceedings of the 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 143–154.

[15] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2009, pp. 44–54.

[16] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads," in *IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2010, pp. 1–11.

[17] Y. Chen, X. Wang, W. Zhu, H. Li, Z. Sun, G. Sun, and Y. Xie, "Access scheme of multi-level cell spin-transfer torque random access memory and its optimization," in *Proceedings of the 53rd IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2010, pp. 1109–1112.

[18] Y. Chen, W.-F. Wong, H. Li, and C.-K. Koh, "Processor caches built using multi-level spin-transfer torque ram cells," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2011, pp. 73–78.

[19] K. Choo, W. Panlener, and B. Jang, "Understanding and optimizing gpu cache memory performance for compute workloads," in *13th International*

Symposium on Parallel and Distributed Computing (ISPDC). IEEE, 2014, pp. 189–196.

[20] G. Chrysos, "Intel xeon phi coprocessor - the architecture," *Intel Whitepaper*, 2014.

[21] L. O. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[22] K. C. Chun, P. Jain, J. H. Lee, and C. H. Kim, "A 3t gain cell embedded dram utilizing preferential boosting for high density and low power on-die caches," *Solid-State Circuits, IEEE Journal of*, vol. 46, no. 6, pp. 1495–1505, 2011.

[23] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better i/o through byte-addressable, persistent memory," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 133–146.

[24] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[25] Z. Diao, Z. Li, S. Wang, Y. Ding, A. Panchula, E. Chen, L.-C. Wang, and Y. Huai, "Spin-transfer torque switching in magnetic tunnel junctions and spin-transfer torque random access memory," *Journal of Physics: Condensed Matter*, vol. 19, no. 16, p. 165209, 2007.

[26] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, 2012.

[27] B. Engel, J. Akerman, B. Butcher, R. Dave, M. DeHerrera, M. Durlam, G. Grynkewich, J. Janesky, S. Pietambaram, N. Rizzo *et al.*, "A 4-mb toggle MRAM based on a novel bit and switching method," *IEEE Transactions on Magnetics*, vol. 41, no. 1, pp. 132–136, 2005.

[28] K. Eshraghian, K.-R. Cho, O. Kavehei, S.-K. Kang, D. Abbott, and S.-M. S. Kang, "Memristor mos content addressable memory (mcam): Hybrid architecture for future high performance search engines," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 19, no. 8, pp. 1407–1417, 2011.

[29] H. Esmaeilzadeh, E. Blem, R. St Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, 2011, pp. 365–376.

[30] P. M. Figueiredo and J. C. Vital, "Low kickback noise techniques for cmos latched comparators," in *Circuits and Systems, 2004. ISCAS'04. Proceedings of the 2004 International Symposium on*, vol. 1. IEEE, 2004, pp. I–537.

[31] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," in *Proceedings*

of the 29th Annual International Symposium on Computer Architecture, 2002, pp. 148–157.

[32] A. Fog, "The microarchitecture of intel and amd cpus," 2008.

[33] N. Goswami, B. Cao, and T. Li, "Power-performance co-optimization of throughput core architecture using resistive memory," in *19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 342–353.

[34] S. Guangyu, "Constructing neural branch prediction with memristive device," Master's thesis, Department of Electrical and Computer Engineering, University of Wisconsin-Madison, May 2011.

[35] S. Hong and H. Kim, "An integrated gpu power and performance model," in *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA)*, 2010, pp. 280–289.

[36] T. Ishigaki, T. Kawahara, R. Takemura, K. Ono, K. Ito, H. Matsuoka, and H. Ohno, "A multi-level-cell spin-transfer torque memory with series-stacked magnetotunnel junctions," in *IEEE Symposium on VLSI Technology (VLSIT)*, 2010, pp. 47–48.

[37] L. Jiang, B. Zhao, Y. Zhang, and J. Yang, "Constructing large and fast multi-level cell stt-mram based cache for embedded processors," in *Proceedings of the 49th Annual Design Automation Conference (DAC)*, 2012, pp. 907–912.

[38] D. A. Jiménez, "Fast path-based neural branch prediction," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2003, pp. 243–252.

[39] D. A. Jimenez, "Piecewise linear branch prediction," in *Proceedings of the 32nd International Symposium on Computer Architecture*. IEEE, 2005, pp. 382–393.

[40] ——, "An optimized scaled neural branch predictor," in *Proceedings of the 29th International Conference on Computer Design*. IEEE, 2011, pp. 113–118.

[41] D. A. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*. IEEE, 2001, pp. 197–206.

[42] N. Jing, Y. Shen, Y. Lu, S. Ganapathy, Z. Mao, M. Guo, R. Canal, and X. Liang, "An energy-efficient and scalable edram-based register file architecture for gpgpu," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 344–355.

[43] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[44] S. H. Jo, K.-H. Kim, T. Chang, S. Gaba, and W. Lu, "Si memristive devices applied to memory and neuromorphic circuits," in *Proceedings of IEEE International Symposium on Circuits and Systems*. IEEE, 2010, pp. 13–16.

[45] A. Jog, A. K. Mishra, C. Xu, Y. Xie, V. Narayanan, R. Iyer, and C. R. Das, "Cache revive: Architecting volatile stt-ram caches for enhanced performance in cmps," in *Proceedings of the 49th Annual Design Automation Conference (DAC)*, 2012, pp. 243–252.

[46] J. Junquera and P. Ghosez, "Critical thickness for ferroelectricity in perovskite ultrathin films," *Nature*, vol. 422, no. 6931, pp. 506–509, 2003.

[47] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *Proceedings of the 28th Annual International Symposium on Computer architecture*, 2001, pp. 240–251.

[48] J. Kin, M. Gupta, and W. H. Mangione-Smith, "The filter cache: an energy efficient memory structure," in *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, 1997, pp. 184–193.

[49] S. Kvatinsky, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Team: Threshold adaptive memristor model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 1, pp. 211–221, 2013.

[50] G. Li, X. Chen, G. Sun, H. Hoffmann, Y. Liu, Y. Wang, and H. Yang, "A stt-ram-based low-power hybrid register file for gpgpus," in *Proceedings of the 52nd Annual Design Automation Conference (DAC)*. ACM, 2015, pp. 103:1–103:6.

[51] Q. Li, M. Zhao, C. J. Xue, and Y. He, "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), 2012, pp. 109–118.

[52] Y. Li, Y. Zhang, H. Li, Y. Chen, and A. K. Jones, "C1c: a configurable, compiler-guided stt-ram l1 cache," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 10, no. 4, p. 52, 2013.

[53] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," *IEEE micro*, no. 2, pp. 39–55, 2008.

[54] M. Mao, W. Wen, Y. Zhang, Y. Chen, and H. Li, "Exploration of gpgpu register file architecture using domain-wall-shift-write based racetrack memory," in *51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[55] S. McFarling, "Combining branch predictors," Technical Report TN-36, Digital Western Research Laboratory, Tech. Rep., 1993.

[56] Y. Meng, T. Sherwood, and R. Kastner, "On the limits of leakage power reduction in caches," in *11th International Symposium on High-Performance Computer Architecture*, 2005, pp. 154–165.

[57] F. Miao, W. Yi, I. Goldfarb, J. J. Yang, M.-X. Zhang, M. D. Pickett, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, "Continuous electrical tuning of the chemical composition of tao x-based memristors," *ACS nano*, vol. 6, no. 3, pp. 2312–2318, 2012.

[58] P. R. Mickel, A. J. Lohn, B. J. Choi, J. J. Yang, M.-X. Zhang, M. J. Marinella, C. D. James, and R. S. Williams, "A physical model of switching dynamics in tantalum oxide memristive devices," *Applied Physics Letters*, vol. 102, no. 22, p. 223502, 2013.

[59] S. Mittal, J. Vetter, and D. Li, "A survey of architectural approaches for managing embedded dram and non-volatile on-chip caches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1524–1537, June 2015.

[60] J. C. Mogul, E. Argollo, M. A. Shah, and P. Faraboschi, "Operating system support for nvm+ dram hybrid main memory." in *HotOS*, 2009.

[61] G. E. Moore *et al.*, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[62] H. Naeimi, C. Augustine, A. Raychowdhury, S.-L. Lu, and J. Tschanz, "Sttram scaling and retention failure," *Intel Technology Journal*, vol. 17, no. 1, p. 54, 2013.

[63] M. S. Papamarcos and J. H. Patel, "A low-overhead coherence solution for multiprocessors with private cache memories," in *ACM SIGARCH Computer Architecture News*, vol. 12, no. 3, 1984, pp. 348–354.

[64] A. Patel, F. Afram, S. Chen, and K. Ghose, "Marss: a full system simulator for multicore x86 cpus," in *Proceedings of the 48th Design Automation Conference*, 2011, pp. 1050–1055.

[65] Y. V. Pershin and M. Di Ventra, "Solving mazes with memristors: A massively parallel approach," *Physical Review E*, vol. 84, no. 4, p. 046703, 2011.

[66] M. D. Pickett, D. B. Strukov, J. L. Borghetti, J. J. Yang, G. S. Snider, D. R. Stewart, and R. S. Williams, "Switching dynamics in titanium dioxide memristive devices," *Journal of Applied Physics*, vol. 106, no. 7, p. 074508, 2009.

[67] M. Poremba, S. Mittal, D. Li, J. S. Vetter, and Y. Xie, "Destiny: A tool for modeling emerging 3d nvm and edram caches," in *Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 1543–1546.

[68] T. Prodromakis, B. P. Peh, C. Papavassiliou, and C. Toumazou, "A versatile memristor model with nonlinear dopant kinetics," *Electron Devices, IEEE Transactions on*, vol. 58, no. 9, pp. 3099–3105, 2011.

[69] J. Rajendran, H. Maenm, R. Karri, and G. S. Rose, "An approach to tolerate process related variations in memristor-based applications," in *24th International Conference on VLSI Desig*. IEEE, 2011, pp. 18–23.

[70] V. J. Reddi, A. Settle, D. A. Connors, and R. S. Cohn, "Pin: a binary instrumentation tool for computer architecture research and education," in *Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture*. ACM, 2004, p. 22.

[71] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the bandwidth wall: Challenges in and avenues for CMP scaling," in *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*, 2009, pp. 371–382.

[72] P. Roy, J. Wang, and W.-F. Wong, "Program analysis for approximation-aware compilation," in *ACM International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, 2015.

[73] M. H. Samavatian, H. Abbasitabar, M. Arjomand, and H. Sarbazi-Azad, "An efficient stt-ram last level cache architecture for gpus," in *51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[74] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, "The eda challenges in the dark silicon era," in *51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2014, pp. 1–6.

[75] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing non-volatility for fast and energy-efficient stt-ram caches," in *IEEE 17th International Symposium on High Performance Computer Architecture (HPCA)*, 2011, pp. 50–61.

[76] G. Snider, R. Amerson, D. Carter, H. Abdalla, M. S. Qureshi, J. Léveillé, M. Versace, H. Ames, S. Patrick, B. Chandler *et al.*, "From synapses to circuitry: using memristive memory to explore the electronic brain," *Computer*, vol. 44, no. 2, p. 21, 2011.

[77] R. St Amant, D. A. Jiménez, and D. Burger, "Low-power, high-performance analog neural branch prediction," in *Proceedings of the 41st IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2008, pp. 447–458.

[78] G. Steven, R. Anguera, C. Egan, F. Steven, and L. Vintan, "Dynamic branch prediction using neural networks," in *Euromicro Symposium on Digital Systems Design*. IEEE, 2001, pp. 178–185.

[79] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-M. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, 2012.

[80] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[81] D. K. Su, M. J. Loinaz, S. Masui, and B. A. Wooley, "Experimental results and modeling techniques for substrate noise in mixed-signal integrated circuits," *IEEE Journal of Solid-State Circuits*, vol. 28, no. 4, pp. 420–430, 1993.

[82] Z. Sun, X. Bi, H. H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, and W. Wu, "Multi retention level stt-ram cache designs with a dynamic refresh scheme," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 329–338.

[83] Z. Sun, W. Wu, and H. Li, "Cross-layer racetrack memory design for ultra high density and low power consumption," in *50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2013, pp. 1–6.

[84] F. Tabrizi, "The future of scalable stt-ram as a universal embedded memory," *Embedded.com*, 2007.

[85] R. Venkatesan, S. G. Ramasubramanian, S. Venkataramani, K. Roy, and A. Raghunathan, "Stag: Spintronic-tape architecture for gpgpu cache hierarchies," in *ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 2014, pp. 253–264.

[86] J. Wang, P. Roy, W.-F. Wong, X. Bi, and H. Li, "Optimizing mlc-based stt-ram caches by dynamic block size reconfiguration," in *32nd IEEE International Conference on Computer Design (ICCD)*, 2014, pp. 133–138.

[87] J. Wang, Y. Tim, W.-F. Wong, and H. H. Li, "A practical low-power memristor-based analog neural branch predictor," in *IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2013, pp. 175–180.

[88] J. Wang, Y. Tim, O. Zhong-Liang, W.-F. Wong, S. Zhenyu, and H. H. Li, "A coherent hybrid sram and stt-ram l1 cache architecture for shared memory multicores," in *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014, pp. 610–615.

[89] J. Wang, X. Dong, and Y. Xie, "Oap: an obstruction-aware cache management policy for stt-ram last-level caches," in *Proceedings of the Conference*

on Design, Automation and Test in Europe (DATE). EDA Consortium, 2013, pp. 847–852.

[90] W. Wen, M. Mao, X. Zhu, S. H. Kang, D. Wang, and Y. Chen, "Cd-ecc: content-dependent error correction codes for combating asymmetric nonvolatile memory operation errors," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. IEEE Press, 2013, pp. 1–8.

[91] W. Wen, Y. Zhang, M. Mao, and Y. Chen, "State-restrict mlc stt-ram designs for high-reliable high-performance memory system," in *Proceedings of the 51st Annual Design Automation Conference (DAC)*, 2014, pp. 1–6.

[92] C. M. Wittenbrink, E. Kilgariff, and A. Prabhu, "Fermi gf100 gpu architecture," *IEEE Micro*, no. 2, pp. 50–59, 2011.

[93] W. Xu, H. Sun, X. Wang, Y. Chen, and T. Zhang, "Design of last-level on-chip cache using spin-torque transfer ram (stt ram)," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 3, pp. 483–493, 2011.

[94] J. J. Yang, M. D. Pickett, X. Li, D. A. Ohlberg, D. R. Stewart, and R. S. Williams, "Memristive switching mechanism for metal/oxide/metal nanodevices," *Nature nanotechnology*, vol. 3, no. 7, pp. 429–433, 2008.

[95] Y. C. Yang, F. Pan, Q. Liu, M. Liu, and F. Zeng, "Fully room-temperature-fabricated nonvolatile resistive memory for ultrafast and high-density memory application," *Nano letters*, vol. 9, no. 4, pp. 1636–1643, 2009.

[96] T.-Y. Yeh and Y. N. Patt, "Two-level adaptive training branch prediction," in *Proceedings of the 24th Annual International Symposium on Microarchitecture (MICRO)*. ACM, 1991, pp. 51–61.

[97] ——, "Alternative implementations of two-level adaptive branch prediction," in *Proceedings of the 19th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 1992, pp. 124–134.

[98] Y. Zhang, L. Zhang, W. Wen, G. Sun, and Y. Chen, "Multi-level cell stt-ram: Is it realistic or just a dream?" in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 526–532.

[99] J. Zhao and Y. Xie, "Optimizing bandwidth and power of graphics memory with hybrid memory technologies and adaptive data migration," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. ACM, 2012, pp. 81–87.

[100] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy reduction for stt-ram using early write termination," in *IEEE/ACM International Conference on Computer-Aided Design-Digest of Technical Papers (ICCAD)*. IEEE, 2009, pp. 264–268.