ENERGY-EFFICIENT MEMORY SYSTEM DESIGN WITH SPINTRONICS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Ashish Ranjan

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2018

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Anand Raghunathan, Chair

School of Electrical and Computer Engineering

Dr. Byunghoo Jung

School of Electrical and Computer Engineering

Dr. Kaushik Roy

School of Electrical and Computer Engineering

Dr. Vijay Raghunathan

School of Electrical and Computer Engineering

Approved by:

Dr. Pedro Irazoqui

Head of the School Graduate Program

To my parents and my siblings for their endless love and encouragement, who always supported me through every path I ever took.

ACKNOWLEDGMENTS

First and foremost, I would like to express my heartfelt gratitude to my advisor Prof. Anand Raghunathan for his constant motivation, inspiration, and numerous insightful suggestions during the entire phase of my PhD life at Purdue. His guidance always helped me steer my research in the right direction and he has been an extraordinary influence in shaping my thought process. He always challenged me to push my boundaries and constantly encouraged me to fearlessly pursue those ideas. Apart from the technical aspects of research, he also made me realize the importance of intuitively communicating and presenting my ideas to others. I am thankful to him for all of his mentoring over the years and hope to carry his lessons in my life beyond graduate school.

Next, I would like to thank Prof. Vijay Raghunathan, Prof. Kaushik Roy and Prof. Byunghoo Jung for their guidance and expert advice throughout my PhD. Their thoughtful suggestions and feedbacks were immensely helpful towards improving my ideas. I am especially thankful to Prof. Vijay Raghunathan for all the brainstorming sessions and productive discussions that provided me deeper insights into my proposals. I express my deepest appreciation for his frequent availability, and his willingness to share his perspective on those ideas.

My sincere thanks to the alumni and current members of Integrated Systems Laboratory – Swagath Venkataramani, Shubham Jain, Vivek Joy Kozhikkottu, Rangharajan Venkatesan, Jacques Pienaar, Shankar Ganesh Ramasubramaniam, Sanchari Sen, Younghoon Kim, and Jacob Stevens, for their support and friendship in the past six years. I truly value all of our discussions, both on academic and personal fronts, and appreciate their numerous insights. I also express my gratitude to my collaborators Arnab Raha, Xuanyao Fong, Zoha Pajouhi, and Mei-Chin Chen with whom I had the opportunity to work on a wide range of projects that helped me develop a broader

perspective to my dissertation. I would also like to thank my friends Mohit Singh, Sambit Mishra and Sayan Basak, who made my stay at Purdue not only enjoyable, but a memorable one.

Special thanks to my parents for their unconditional love and their unwavering support through every step of my PhD journey. This journey wouldn't have been possible without their encouragement, and I sincerely thank them for their endless patience. To my siblings, Anay Raj, Amit Wats, and Smita Bhushan, and my sister-in-law Rani Roy, I can't thank all of you enough, for inspiring me to always chase my dreams. I am especially grateful to my brother Anay Raj for always believing in me and constantly motivating me to deliver my best. The following pages of my thesis are solely dedicated to all of them for their love and sacrifices.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

ABSTRACT

Ranjan, Ashish Ph.D., Purdue University, December 2018. Energy-efficient Memory System Design with Spintronics. Major Professor: Anand Raghunathan.

Modern computing platforms, from servers to mobile devices, demand ever-increasing amounts of memory to keep up with the growing amounts of data they process, and to bridge the widening processor-memory gap. A large and growing fraction of chip area and energy is expended in memories, which face challenges with technology scaling due to increased leakage, process variations, and unreliability. On the other hand, data intensive workloads such as machine learning and data analytics pose increasing demands on memory systems. Consequently, improving the energy-efficiency and performance of memory systems is an important challenge for computing system designers.

Spintronic memories, which offer several desirable characteristics – near-zero leakage, high density, non-volatility and high endurance – are of great interest for designing future memory systems. However, these memories are not drop-in replacements for current memory technologies, *viz.* Static Random Access Memory (SRAM) and Dynamic Random Access Memory (DRAM). They pose unique challenges such as variable access times, and require higher write latency and write energy. This dissertation explores new approaches to improving the energy efficiency of spintronic memory systems.

The dissertation first explores the design of *approximate memories*, in which the need to store and access data precisely is foregone in return for improvements in energy efficiency. This is of particular interest, since many emerging workloads exhibit an inherent ability to tolerate approximations to their underlying computations and data while still producing outputs of acceptable quality. The dissertation pro-

poses that approximate spintronic memories can be realized either by reducing the amount of data that is written to/read from them, or by reducing the energy consumed per access. To reduce memory traffic, the dissertation proposes *approximate memory compression*, wherein a quality-aware memory controller transparently compresses/decompresses data written to or read from memory. For broader applicability, the quality-aware memory controller can be programmed to specify memory regions that can tolerate approximations, and conforms to a specified error constraint for each such region. To reduce the per-access energy, various mechanisms are identified at the circuit and architecture levels that yield substantial energy benefits at the cost of small probabilities of read, write or retention failures. Based on these mechanisms, a quality-configurable Spin Transfer Torque Magnetic RAM (STT-MRAM) array is designed in which read/write operations can be performed at varying levels of accuracy and energy at runtime, depending on the needs of applications. To illustrate the utility of the proposed quality-configurable memory array, it is evaluated as an L2 cache in the context of a general-purpose processor, and as a scratchpad memory for a domain-specific vector processor.

The dissertation also explores the design of caches with Domain Wall Memory (DWM), a more advanced spintronic memory technology that offers unparalleled density arising from a unique tape-like structure. However, this structure also leads to serialized access to the bits in each bit-cell, resulting in increased access latency, thereby degrading overall performance. To mitigate the performance overheads, the dissertation proposes a reconfigurable DWM-based cache architecture that modulates the active bits per tape with minimal overheads depending on the application's memory access characteristics. The proposed cache is evaluated in a general purpose processor and improvements in performance are demonstrated over both CMOS and previously proposed spintronic caches.

In summary, the dissertation suggests directions to improve the energy efficiency of spintronic memories and re-affirms their potential for the design of future memory systems.

1. INTRODUCTION

The proliferation of several applications ranging from recognition, vision, machine learning, search, data-driven inference, information analytics *etc.,* has resulted in an explosion in the creation and consumption of various forms of digital data across the entire spectrum of computing platforms from mobile and wearable devices to servers [1, 2]. These workloads process large amounts of data, which require real-time analytics to derive value from them, placing significant demands on the memory subsystems. Moreover with every processor generation, we are also noticing a growing trend in the number of processor cores executing these applications. Feeding data to 10s-1000s of such cores requires even larger memories. Fig. 1.1 illustrates the growing trend observed in both on-chip and off-chip memory capacity over the years, across different computing platforms including general purpose processors, System-on-Chips (SoCs) and GPUs. Specifically, if we observe the cache trends in Intel processors (top left in Fig. 1.1), both the absolute number of cache transistors and the fraction of chip area dedicated to caches is increasing with each processor generation. Similar trends have been observed for on-chip memories in GPUs, and SoCs. Fig. 1.1 (bottom right) also presents the unabated growth observed in DRAM capacity over the years.

CMOS based memory technologies, *viz.* SRAM, embedded-DRAM, and DRAM, have been the workhorses of memory design for several decades. The continued scaling of transistor dimensions and supply voltage in these memory technologies have led to higher density (offering larger capacity for memories), lower dynamic power consumption and higher performance. However, these memory technologies face growing design challenges such as increased process variations, higher leakage energy, increased unreliability *etc.*, in the nanometer regime as CMOS scales towards its fundamental limits. These design challenges along with an ever-increasing demand for memories

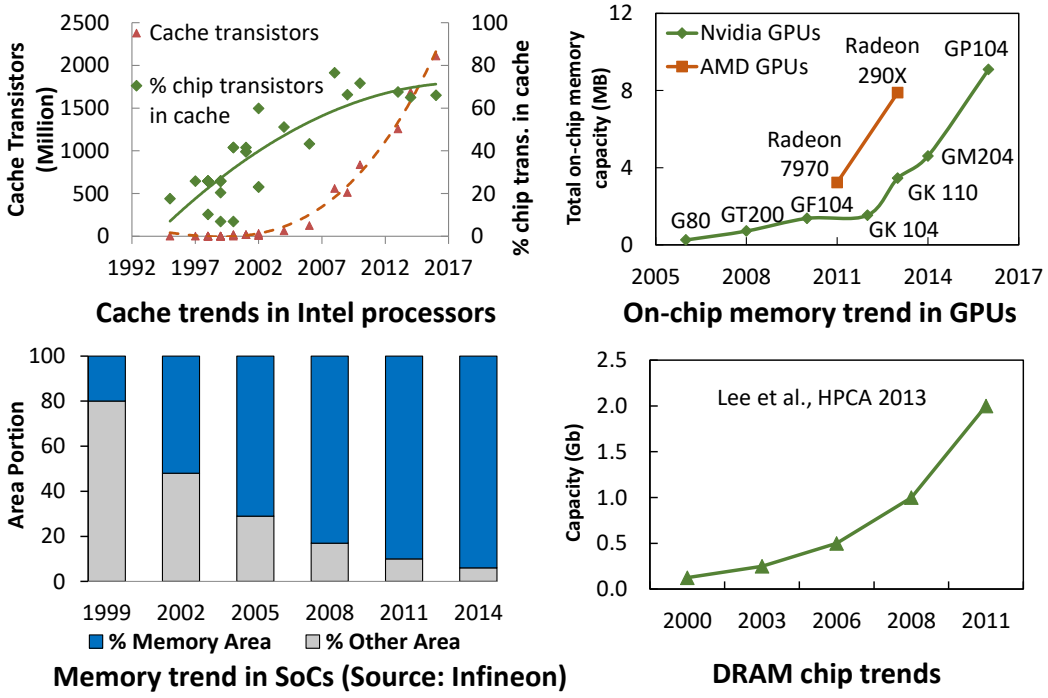


Fig. 1.1.: Memory capacity trends across computing platforms

have fueled the quest for alternative memory technologies that can either complement or potentially replace the current memory systems.

Several memory technologies such as Phase Change Memory (PCM), Ferroelectric RAM (FeRAM) *etc.* have emerged as promising candidates in this quest for potential CMOS replacements. Spintronic memories are one such emerging alternative that have demonstrated great potential and garnered significant interest in recent years, with several prototype demonstrations and early commercial offerings [3, 4]. Unlike traditional charge-based devices, spintronic devices manipulate the spin orientation of electrons in a ferromagnetic material to represent and process information. These devices possess several favorable characteristics that are beneficial to memory design: (i) non-volatility, that results in ultra-low leakage energy, (ii) high density, that enables larger memory capacity, and (iii) high endurance. However, these memories have certain limitations, such as higher write latency and write energy *etc.*, owing to the fundamentally different storage and switching mechanisms that these memory

technologies employ. Therefore, they are not drop-in replacements for current CMOS memory technologies. This dissertation explores suitable optimizations at the circuit and architecture-level for spintronic memories that exploit the strengths of these memories while mitigating their drawbacks.

The following sections present an overview of the current scaling challenges in CMOS technology, a summary of the spintronic memory technologies that have shown great promise in recent years, and a brief description of the thesis contributions.

1.1 CMOS Scaling Challenges

Moore's law of transistor scaling for CMOS technology has resulted in ever-increasing transistor performance, higher density and lower energy consumption. However, continued technology scaling in the deep nanometer regime has presented several roadblocks such as higher leakage energy, increased process variations, new reliability issues, *etc.*

1.1.1 Higher leakage and power density

As transistors continue to scale down into the nanometer regime, the leakage energy of these transistors has been rapidly increasing. Traditionally, the leakage in a CMOS transistor is dominated by sub-threshold static leakage current from the source, gate-oxide leakage current and gate induced drain leakage current (GIDL). Each of these components have exponentially increased over the years. Sub-threshold leakage current(I_{sub}), a function of the inverse of the threshold voltage (V_T), is increasing due to shorter effective channel length and longer transistor width with every technology generation. Similarly, the tunneling current flowing through the gate is on the rise, due to aggressive reduction in the gate oxide thickness. Fig. 1.2 summarizes the increasing trend in leakage power witnessed over the years. As shown in figure, the leakage power contributes a substantial fraction ($\sim$40%) of the total power consumption in modern computing platforms.

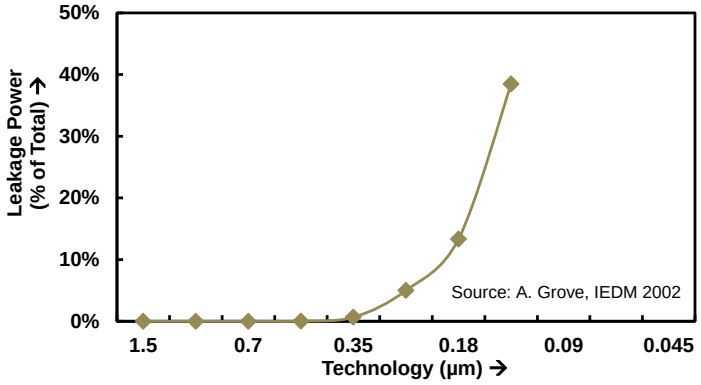


Fig. 1.2.: Fraction of leakage power with different technology generations

Another major challenge to the scaling of CMOS transistors is the growing power density of computing systems. This is primarily due to an increased number of transistors per chip without a proportional decrease in power per transistor. Fig. 1.3 shows the increasing trends in transistor count and clock frequency for various Intel microprocessors. With the decline of "classical" scaling of supply voltage (also known as Dennard scaling) due to increased short channel effects, the power density on the chip no longer remains constant. The slow down in supply voltage scaling implies an increase in the power density of the chip with each generation. This eventually results in higher on-chip temperature, mandating the need for new cooling mechanisms to mitigate these concerns. Eventually, these trends led to the saturation of clock frequencies and the use of multi-cores or parallelism to achieve performance improvements.

1.1.2 Increased process variations

Process variations mainly arise due to limitations of the chip fabrication process, *i.e.*, lithographic process, and often cause key process parameters such as effective channel length, gate oxide thickness, *etc.* to drift from their designed values. Consequently, these variations can impact several important transistor characteristics such as threshold voltage, causing the switching characteristics of transistors to vary

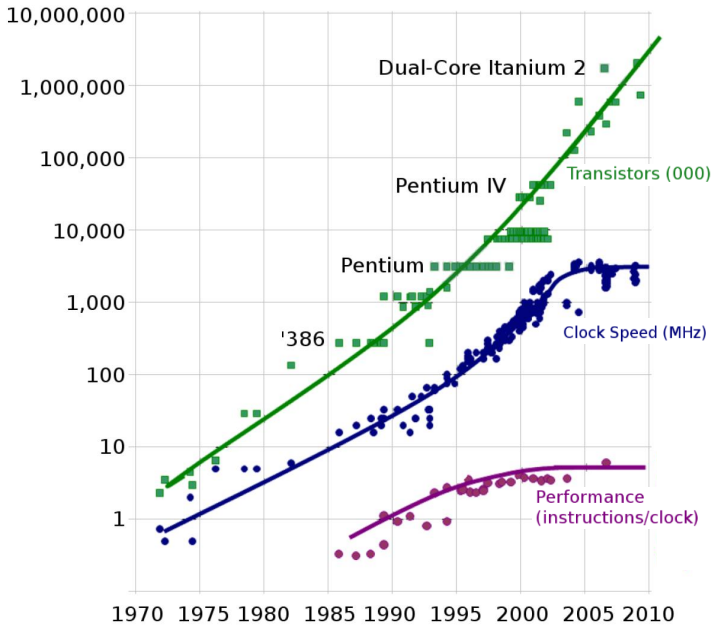


Fig. 1.3.: Transitor count and clock frequency trend in Intel microprocessors [5]

substantially. As CMOS transistors scale to the nanometer regime, the impact of variations is becoming higher. Fig. 1.4 demonstrates the impact of process variations on the ON and OFF currents of nearly identical transistors in a modern CMOS process technology node. As shown in the figure, the ON current varies by over 2× across transistors manufactured in the same process technology while the leakage (OFF) current drifts by almost 100×.

The increase in process variations has mandated designers to over-design their systems such that they operate correctly under different conditions of variations. With shrinking transistor dimensions, the corresponding guard-band required to mitigate the impact of variations is also increasing. This has resulted in a growing concern across the semiconductor industry that the additional guard-bands introduced in the chip design process could eventually negate all the benefits of transistor scaling.

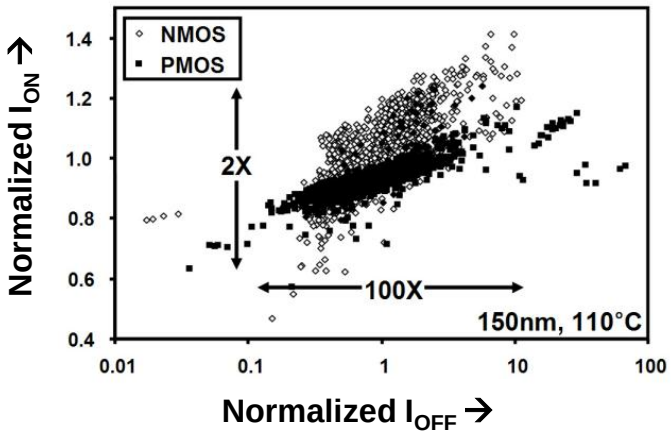


Fig. 1.4.: I_{ON} and I_{OFF} variation for 150nm technology (Source: Intel)

1.1.3 Reliability concerns

There are several physical failure mechanisms that can impact the reliability of CMOS transistors. Some of them include Hot Carrier Damage (HCD), Time Dependent Dielectric Breakdown (TDDB), Negative Bias Temperature Instability (NBTI), *etc.* HCD and NBTI result in drastic variation in the transistor threshold voltage over prolonged periods. On the other hand, TDDB causes wearing-out of the insulating properties of the CMOS transistor's gate, which eventually leads to a conducting path between the gate and the CMOS substrate.

The continued scaling of transistor dimensions has further led to an increase in the above discussed physical failures thereby degrading the reliability of these transistors.

Over the years, the fraction of chip area occupied by memories is steadily increasing due to increasing data set sizes and growing number of cores. Thus, the above discussed scaling challenges translate to diminishing benefits with every technology node, presenting a major roadblock to continued scaling of CMOS memories.

1.2 Emerging Memory Technologies

Many research efforts have been aimed at finding alternate memory technologies that can address the limitations of traditional CMOS memories in recent years. Several promising candidates have emerged from this quest. The following paragraphs provide a brief description of some of the promising memory technologies and also present a comparison of their key characteristics.

1.2.1 Spin Transfer Torque Magnetic RAM (STT-MRAM)

STT-MRAM is a non-volatile memory technology that manipulates the spin orientation of electrons in a ferro-magnetic material such as CoFeB to represent and process data. Fig. 1.5 shows the structure of a standard STT-MRAM bit cell. It consists of an access transistor and a magnetic tunnel junction (MTJ), which is in turn composed of a pinned layer and a free layer separated by a tunneling oxide (*i.e.* MgO). The pinned layer has a fixed magnetization while the free layer can be programmed to change its magnetization orientation. The relative orientation of the free layer and the pinned layer determines the logic state of the data stored in the bit-cell (assuming logic "0" when parallel, *i.e.* low resistance and "1" when anti-parallel, *i.e.* high resistance). STT-MRAMs possess several characteristics that favor memory design: (i) ultra-low leakage energy as a consequence of non-volatility, (ii) high density, and (iii) high endurance or life cycle. Although STT-MRAMs have near-zero leakage energy compared to CMOS memories, their overall energy efficiency is still limited by read and write operations. Reads are bottlenecked by the need to perform single-ended sensing, while the commonly used write mechanism of spin transfer torque switching requires large currents, leading to energy inefficient writes.

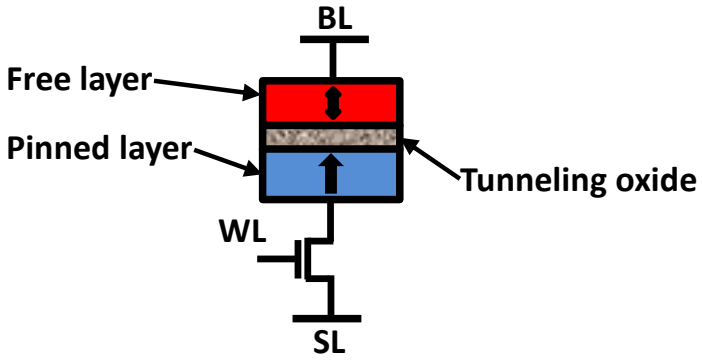


Fig. 1.5.: STT-MRAM bit-cell

1.2.2 Domain Wall Memory (DWM)

DWM is an advanced spintronic memory technology which is highly promising due to its unparalleled density when compared to other spintronic memories such as STT-MRAM [6,7]. DWMs have a unique tape like structure that achieves very high density by packing several (∼10-100) bits into the domains of a ferromagnetic nanowire [6], as shown in Fig. 1.6. A key feature of DWMs is that the bits stored in the nanowire can be shifted by applying a current pulse. However, this structure also introduces serialized accesses to the bits in each bit-cell via shift operations, resulting in higher access latency. For instance, our evaluations show that the performance bottleneck due to the shift operations can be as high as 37% for an iso-area DWM-based cache compared to a 2MB SRAM-based cache in 32nm technology.

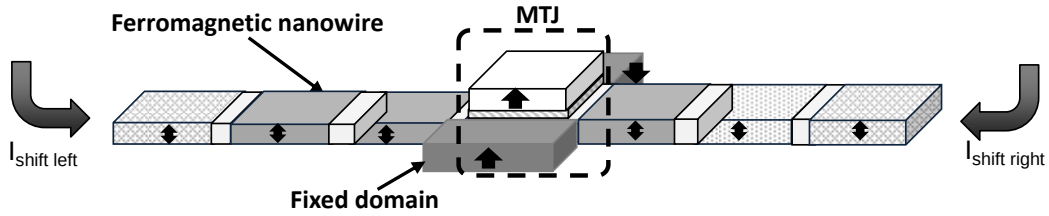


Fig. 1.6.: Domain Wall Memory

1.2.3 Comparison of different memory technologies

Fig. 1.7 shows a comparison of some of the key metrics for different memory technologies based on [6]. Spintronic memories, *i.e.* STT-MRAM and DWM possess several desirable features such as near-zero leakage power, high density and access latencies closest to SRAM and DRAM memories, thereby making them lucrative for future memories. Notably, DWM offers extremely high density and outperforms even other emerging memory technologies such as PCM and FeRAM *etc.* Moreover, these spintronic memories have much smaller access latencies compared to other traditional non-volatile memory technologies such as flash memories. On the other hand, the write energy (and write latency) of these spintronic memories is substantially higher than SRAM and DRAM, posing a major challenge to the design of future spintronic memory subsystem. For example, our evaluations reveal that the write latency for an iso-capacity spintronic cache can be as high as 2.2× compared to a 2MB SRAM-based cache in 45nm technology.

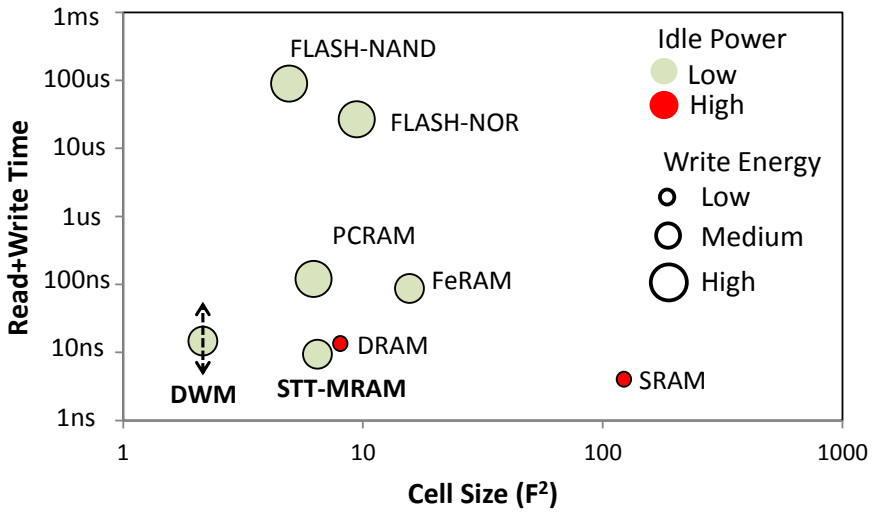


Fig. 1.7.: Comparison of emerging memory technologies [6, 8]

1.3 Thesis Contributions

Many emerging applications that drive the demand for memory also exhibit *intrinsic resilience*, *i.e.*, an ability to tolerate approximations in the underlying computations or data while still producing results of acceptable quality [9]. This intrinsic resilience stems from several factors as shown in Fig. 1.8:

- The algorithms are designed to handle noisy/redundant real-world inputs, and therefore are inherently robust to errors in the underlying computations.
- The computation patterns in these applications are often statistical and iterative in nature where errors tend to self-heal (or cancel) over time with multiple iterations.
- The usage model of many applications is such that the user is conditioned to accept less-than-perfect outputs, or a range of outputs is considered identical due to a lack of golden answer.

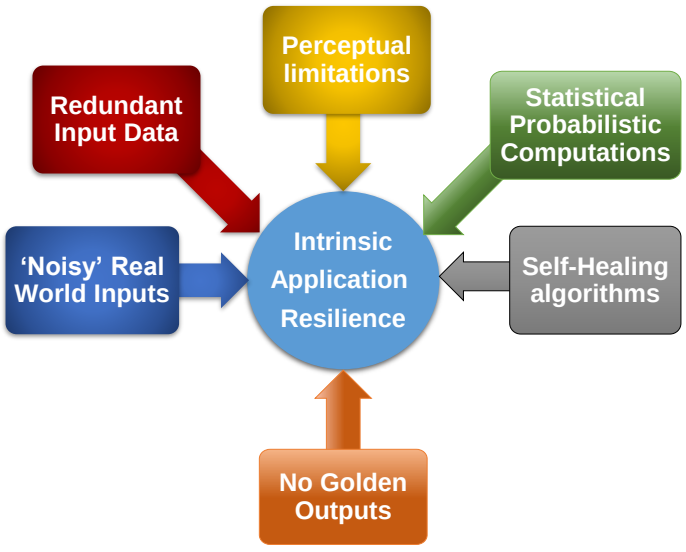


Fig. 1.8.: Sources of intrinsic application resilience [10]

1.3.1 Approximate Memory Subsystem

Approximate computing exploits this resilience to improve energy and performance through techniques at various levels of the computing system stack [10, 11]. Adopting this approach, the dissertation explores the design of *approximate spintronic memories*, wherein the need to store and access data precisely is relaxed for improvements in energy efficiency. Fig. 1.9 provides a high-level overview of an approximate memory system. The key idea behind the approach is to either (i) reduce the amount of data that is written to/read from these memories through approximate memory compression, or (ii) reduce the energy consumed per read/write access using a quality-configurable memory design. The following paragraphs briefly describe the two approaches.

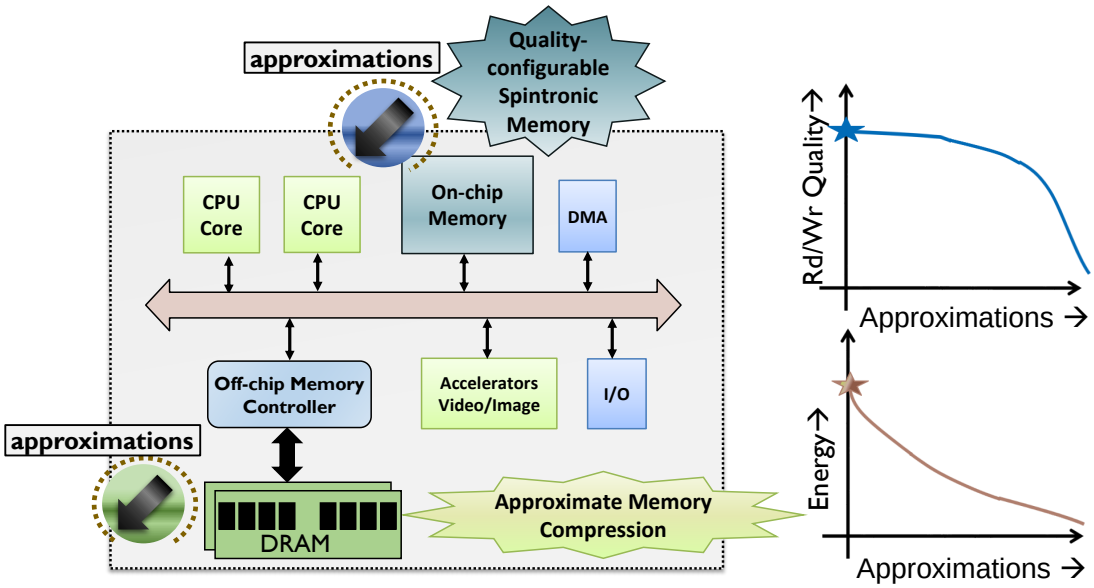


Fig. 1.9.: Approximate memory system: Concept

Approximate Memory Compression

The thesis presents *approximate memory compression*, a technique that leverages the intrinsic resilience of emerging workloads to reduce memory traffic. To realize

approximate memory compression, the memory controller is enhanced to be aware of memory regions that contain approximation-resilient data, and to transparently compress/decompress the data written to/read from these regions. To provide control over approximations, the quality-aware memory controller conforms to a specified error constraint for each approximate memory region. The proposed quality-aware memory controller is exposed to the application by: (i) a software interface that can identify data structures that are resilient to approximations, and (ii) a runtime quality control framework that automatically determines the error constraints for the identified data structures such that a given target application-level quality is maintained. To demonstrate the feasibility of the proposed concepts, a hardware prototype was also implemented using the Intel UniPHY-DDR3 memory controller and Nios-II processor, a Hynix DDR3 DRAM module, and a Stratix-IV FPGA development board. Across a wide range of machine learning benchmarks, approximate memory compression on average obtains 2.0× benefit in main memory energy for an STT-MRAM based DDR3 memory. It also achieves 9.3% improvement in execution time for a small (0.3%) loss in output quality.

Approximate Memory Design with STT-MRAMs

To improve the read and write energy efficiency of STT-MRAMs, the dissertation identifies a combination of different approximation techniques at the circuit and architecture levels that yield significant energy benefits for small probabilities of errors in reads, writes, and retention. Based on these mechanisms, a quality-configurable memory array is designed in which data can be stored to varying levels of accuracy depending on the application requirements.

The proposed array is evaluated in two different memory designs: (i) STAxPad, a scratchpad in the memory hierarchy of a domain-specific vector processor, and (ii) STAxCache, a last-level cache architecture in a general purpose processor, that retains the full flexibility of a conventional cache, while allowing for different levels of

approximation to different parts of a program's memory address space. To expose STAxPad to applications, quality-aware load/store instructions are introduced within the ISA of the vector processor. The thesis also introduces a simple interface that allows the programmer to specify the quality requirements for different data structures, and new instructions in the ISA to expose this information to STAxCache.

The thesis adopts a device-to-architecture modeling framework to evaluate the proposal. Our experiments reveal 1.67× improvement in STAxPad energy and 1.44× improvement in STAxCache energy for negligible ($< 0.5\%$) loss in application quality across a range of machine learning benchmarks.

1.3.2 Reconfigurable Cache Architecture with DWMs

The dissertation also explores the design of caches with DWMs. A key challenge to performance of a DWM-based cache is the inherent tape-like structure of DWM that leads to serialized access to the bits stored in each bit-cell, resulting in increased access latency. Prior efforts address this challenge either by limiting the number of bits per tape, in effect sacrificing the density benefits of DWM, or through cache management policies that can only partly alleviate the shift overhead.

This thesis makes a key observation that there exists significant heterogeneity in sensitivity to cache capacity and access latency across different applications, and across distinct phases of an application. Moreover, the DWM tapes offer a natural mechanism to trade-off density for access latency by limiting the number of domains of each tape that are actively used to store cache data. Based on this insight, the thesis presents DyReCTape, a dynamically reconfigurable cache that packs maximum bits per tape and leverages the intrinsic capability of DWMs to modulate the active bits per tape with minimal overhead. DyReCTape uses a history-based reconfiguration policy that tracks the number of shift operations incurred and miss rate to appropriately tailor the capacity and access latency of the DWM cache. The thesis further proposes two performance optimizations to DyReCTape: (i) a lazy

migration policy to mitigate the overheads of reconfiguration, and (ii) re-use of the portion of the cache that is unused (due to reconfiguration) as a victim cache to reduce the number of off-chip accesses. We evaluate DyReCTape using applications from the PARSEC and SPLASH benchmark suites. Our experiments demonstrate that DyReCTape achieves 19.8% performance improvement over an iso-area SRAM cache and 11.7% performance improvement (due to a 3.4× reduction in the number of shifts) over a state-of-the-art DWM cache.

1.4 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 provides a brief overview of the previous research efforts related to memory design with emerging technologies and presents the thesis contributions in their context. Chapter 3 provides background information related to the structure and operation of two spintronic memory technologies, *viz.*, STT-MRAM and DWM. Chapter 4 describes the concept of approximate memory compression, the design of a quality-aware memory controller, and the proposed software framework for its broader applicability. Chapter 5 presents the various mechanisms to reduce per-access energy for spintronic memories, and details two different memory designs, *viz.*, STAxPad and STAxCache. Chapter 6 presents DyReCTape, the dynamically reconfigurable DWM-based cache architecture. Finally, Chapter 7 concludes this dissertation with a brief description of key directions for future research.

2. RELATED WORK

Many emerging memory technologies such as PCM, STT-MRAM *etc.*, have been well researched over the years to replace CMOS memories. Several research efforts have demonstrated the potential of STT-MRAM and PCM by realizing them as lower level cache and main memory respectively [12–20]. While these emerging memory technologies promise high density and near-zero leakage, they also have severe limitations such as high write energy and high write latency. Furthermore, PCMs also suffer from limited endurance problem. Several techniques have been proposed at the device, circuit and architecture level to mitigate these drawbacks. Over the years few other technologies such as domain wall memory have also attracted significant interest as a promising candidate for replacing CMOS-based memories [6, 8, 21–26]. In this chapter, we describe some of the research efforts that employ these emerging technologies in the context of both on-chip and off-chip memories.

2.1 Spin Transfer Torque Magnetic RAM (STT-MRAM)

Prior research efforts have explored techniques spanning devices, circuit and architecture level in order to achieve energy-efficient STT-MRAMs. We next provide an overview of some of these techniques at the various levels of design abstraction.

2.1.1 Device and circuit techniques

At the device level, researchers have proposed different device structures such as dual-pillar MTJ, tilted MTJ, dual-barrier MTJ *etc.*, in order to address the write-inefficiency of STT-MRAMs [27–29]. At the circuit level, schemes such as 2T-1R bit-cell with dual source line, bit-line voltage clamping, co-design of bit-cell access

transistor and the supply voltage *etc.* have been proposed to improve energy efficiency of STT-MRAMs [30–32]. Few other research efforts have analyzed the impact of process variations and also focused on improving the read latency with efficient read sensing schemes with STT-MRAMs [30, 33, 34]. Besides, [35] proposed multi-level STT-MRAMs in order to enhance the bit-cell density.

2.1.2 Architectural techniques

At the architecture level, [12, 36, 37] addressed the write-inefficiency of STT-MRAMs using a hybrid CMOS-spintronic cache organization that selectively directs write-intensive memory blocks to the CMOS memory, while retaining the remaining blocks in STT-MRAM memory. Another approach focused on eliminating redundant writes to memory, either by comparing the previously stored data before performing writes [38] or by tracking dirty data at a finer granularity [39], in order to minimize the write energy. A third approach proposed volatile spintronic memories [40, 41], wherein the non-volatility of STT-MRAMs was relaxed for smaller write energy and suitable refresh techniques were proposed to avoid data retention errors. Some other efforts [42, 43] have utilized dynamic reconfiguration of hybrid caches to address the write inefficiencies of STT-MRAMs, whereas [35] proposed a set-remapping scheme as a mechanism to achieve energy-efficient encoding of bits to various resistance levels in a multi-level STT-MRAM-based cache. In order to address the write latency of STT-MRAMs, [44] proposed the use of read-preemptive write buffers.

2.2 Phase Change Memory (PCM)

In this section, we focus our discussion to prior works that utilize PCMs for memories.

2.2.1 Device and circuit techniques

At the device level, various structures such as μ-trench, wall, cross spacer, edge, *etc.* have been proposed to mitigate the high write current for PCMs [45–50]. In order to address the limited endurance problem, [51] proposed adding suitable amount of doping material to the phase change material. At the circuit level, [52] proposed fine-grained current regulation and voltage upscaling to lower the RESET current and consequently improve the overall lifetime of PCMs.

2.2.2 Architectural techniques

In order to address the limited endurance, [53–55] proposed lowering the number of bits written to phase change memory through suitable architectural schemes. [53] proposed eliminating redundant bit-writes by comparing the new value to be written against the value stored in the cell, while [54] proposed further enhancements to it by either writing the new data word or the "flipped" value such that it increases the redundant bits. In [55], the authors explored static and dynamic profiling methods to identify frequent data blocks written to the PRAM memory and subsequently used existing compression techniques to minimize the write intensity of those blocks. Another approach to address the lower endurance was proposed in [18] where the authors utilized different wear-leveling policies to spread out the write intensity evenly among all the PCM cells. On the other hand, in order to mitigate the high write energy for PCMs, [56] proposed the concept of approximate memories that consume lower write energy but at the cost of errors. Specifically, it proposed the following: (i) lowering the number of programming pulses used to write PCMs, trading-off write errors for energy, and (ii) using worn-out blocks that have exhausted error correction resources by performing partial error correction on the most significant bits.

2.3 Domain wall memory (DWM)

Domain wall memory (DWM) is a spintronic memory technology that offers very high density and improved energy efficiency compared to STT-MRAM, PCM and other emerging memory technologies. This has kindled great interest in using DWMs to realize caches both in the context of general purpose processors [8, 23, 24, 26, 57] and domain specific accelerators such as GPUs [21,25,58]. In this section, we describe the prior research efforts related to DWMs.

2.3.1 Device and circuit level

DWM was first explored in the context of secondary storage [6] mainly due to its promising density. Considering its attractive benefits of both density and energy, researchers have investigated DWMs at the device and circuit level [22–24, 59, 60]. Multiple prototypes of DWM have also been demonstrated [61, 62]. [22, 59] proposed physics-based models that capture the domain wall dynamics, whereas [60] proposed a multi-level cell that uses domain wall magnets in order to improve the read/write performance, density and write energy of traditional spin-based memories.

2.3.2 DWM Architectures

Recent efforts [8, 23–26, 57] have explored the applicability of DWM for on-chip memories. [8,23,24,57] explored DWM-basedcaches in the context of general purpose processors with fixed numbers of bits on a tape and proposed cache management policies to mitigate the performance penalty due to serialized memory access. Subsequently, [26] presented several layout strategies along with way-based mapping and resizing for DWM-based cache architecture. In [63], the authors analyzed the benefits of DWM-based main memory. In addition, DWMs have also been explored in the context of graphics processors [25, 58] and accelerators [21]. [21] proposed the use of DWMs to realize a FIFO for recognition and mining processor, while [58] pre-

sented novel architectural schemes such as register remapping for GPGPU register files in order to overcome the shift latency. [25] explored DWMs in order to realize on-chip caches for GPGPUs and proposed techniques such as warp-based prediction to address their variable access latencies.

2.4 Thesis contributions

The primary contributions of this dissertation are different from and complementary to the earlier efforts in the following aspects:

Approximate memory compression: In the context of CMOS-based on-chip memories, [64] proposed specific extensions to the load/store queue for handling variable precision across the memory hierarchy. [65,66] explored a cache architecture that stores similar data values at the same cache location, thereby increasing the effective cache capacity, whereas [67] proposed skipping cache loads on misses to mitigate the miss penalty. [68] explored voltage scaling to achieve leakage energy benefits in cache. All these efforts require significant changes to the processor or the cache hierarchy. In contrast, our proposal transparently applies to all components that use the memory controller to access memory, making it applicable across a broad class of computing platforms.

For DRAMs, [69–73] proposed techniques such as modulating the refresh rate to obtain energy benefits at the cost of retention errors. These techniques target refresh/idle energy and are complementary to our proposal, which focuses on reducing the overall read/write energy.

Approximate memory design for energy-efficient spintronic memories: Prior efforts have explored approximations for both off-chip and on-chip memories. [69, 72] proposed relaxing refresh rate for DRAMs to reduce refresh power at the cost of retention errors. On the other hand [56] proposed (i) lowering the number of programming pulses used to write phase change memories, trading-off write errors for

energy, and (ii) using worn-out blocks that have exhausted error correction resources by performing partial error correction on the most significant bits.

In the context of CMOS on-chip memories, [68] proposed supply voltage modulation per cache way to obtain leakage benefits at the cost of failures. [74] explored scaling the supply voltage in an application-specific hybrid SRAM array to achieve energy benefits at the cost of errors. [65] explored a complementary approach of storing similar values as a single cache block to reduce the overall cache energy, whereas [67] proposed skipping cache loads on a miss to lower the cache miss penalty.

Our work differs from these efforts on multiple key fronts. First, a large majority of the above approximation mechanisms are not directly applicable to STT-MRAMs, requiring the exploration of new mechanisms that manifest specific energy-error trade-offs unique to STT-RAMs. Second, our proposal of quality-configurable memory arrays is unique in that it allows the quality of read/write operations to be controlled by specifying acceptable error constraints for groups of bits within the word. This allows us to regulate the numerical significance of the errors incurred during approximate memory operations, enabling better control over the quality-energy trade-off. Third, unlike most of the efforts [56, 69] that involve software directly managing approximations to these memories through load/store instructions or type qualifiers, our work also explores approximate storage in an STT-MRAM cache requiring the cache hardware to dynamically regulate the quality of memory accesses. This involves fundamentally different trade-offs because the same cache region often requires different data storage accuracies at different times. Finally, in contrast to [68] that reduces the cache capacity visible to both accurate and resilient data by enforcing a way-based quality, we perform quality regulation per cache line without limiting associativity.

Reconfigurable cache design using DWM tapes: Reconfigurable caches have been extensively researched in the context of CMOS-based caches. Most of the efforts have targeted optimizing energy consumption with minimal impact on system performance. [75] proposed dynamically reconfiguring the associativity of caches, while [76] used associativity, capacity and line size reconfiguration for achieving energy effi-

ciency. Our work explores a dynamically reconfigurable architecture to address the unique challenge posed by DWM-based caches, *i.e.*, performance penalty due to shifts. The unique structure of DWM provides a natural knob to vary cache capacity and latency at run-time by modulating the bits per tape. However, as shown in our results, such a scheme also requires a suitable reconfiguration policy and optimizations that mitigate the reconfiguration overheads in order to achieve improved performance.

3. BACKGROUND

This chapter provides relevant background information on two spintronic memory technologies – STT-MRAM and DWM, that have been investigated in this dissertation.

3.1 STT-MRAM

Fig. 3.1 (inset) shows the structure of a standard STT-MRAM bit cell. It consists of an access transistor and a storage element, *i.e.* magnetic tunnel junction (MTJ). MTJ is composed of a pinned layer and a free layer separated by a thin tunneling oxide. The pinned layer has a fixed magnetization while the magnetic orientation of the free layer can be varied using spin-polarized current. The relative orientation of the free layer and the pinned layer determines the resistance offered by the MTJ, which in turn represents the logic state of the data stored in the bit-cell. In our case, we assume logic "0" with low resistance state when the two layers are parallel and "1" when they are anti-parallel.

3.1.1 Read and write operation

A read operation is performed by enabling the word line (WL), applying a bias voltage ($<< V_{DD}$) from the bit line (BL) to source line (SL), and sensing the current flowing through the MTJ. This current is compared against a global reference value to determine whether the logic state is high or low. If the current is higher (lower) than the reference, the MTJ is in the low (high) resistance state, and hence the read value is a "0" ("1"). The read current is typically substantially lower than the critical switching current of the MTJ so that the logic state stored in the MTJ is not disturbed

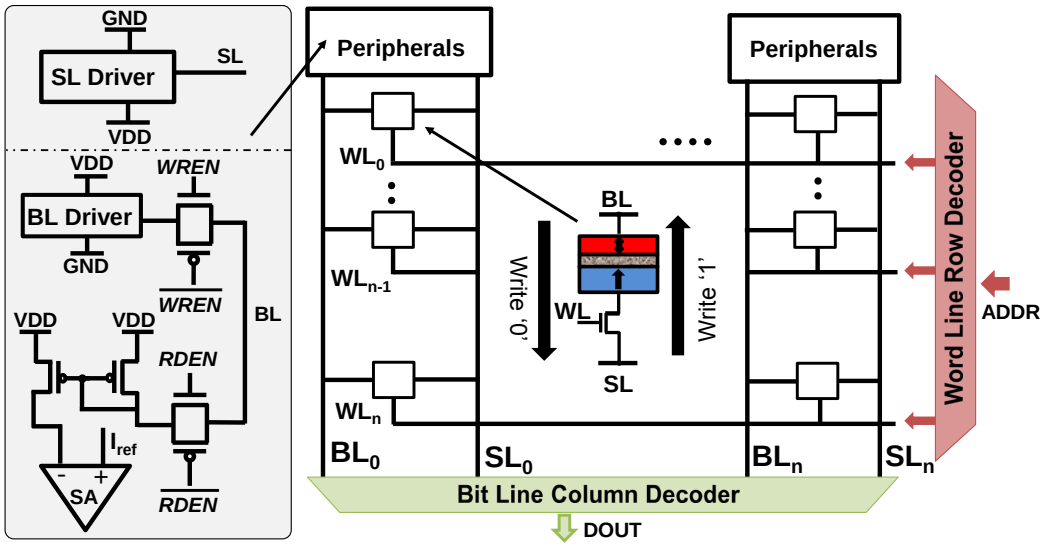


Fig. 3.1.: STT-MRAM bit-cell array

during the read operation. During the write operation, a current greater than the critical switching current is passed through the bit-cell. The current direction (also shown in Figure 3.1) is determined by the logic value to be written into the bit-cell. MTJs require increasing currents for lower switching durations [77]. Typically, the large switching currents required to achieve acceptable write latencies poses a major challenge in the design of energy efficient on-chip spintronic memories.

3.1.2 STT-MRAM array

Fig. 3.1 shows a portion of an STT-MRAM memory array along with its peripheral circuitry. A write driver (BL driver) or read biasing circuit with sense amplifier are selected by an analog mulitplexer to drive each bit line for write and read operations, respectively. The row decoder is used to turn on the appropriate word line, whereas the column decoder turns on the appropriate bit and source lines, depending on the address being accessed.

3.2 Domain Wall Memory

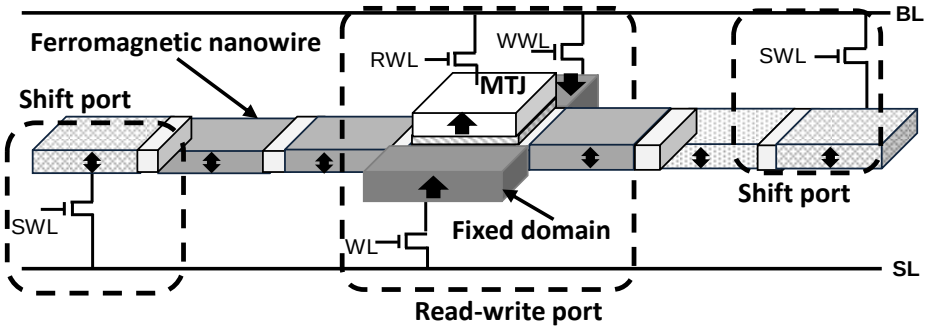


Fig. 3.2.: Multi-bit DWM cell

Fig. 3.2 shows the structure of a multi-bit DWM cell consisting of a ferromagnetic nanowire, a read/write port, and shift ports. The ferromagnetic wire consists of multiple free domains, each of which can be programmed to store a bit. The read/write port is made up of a magnetic tunneling junction (MTJ), two fixed domains, and 3 access transistors. The MTJ is formed by a free domain of the nanowire and a fixed ferromagnet, separated by a tunneling oxide, and is used to sense the data stored in the free domain during the read operation. The fixed domains have opposite spin polarizations and are used to write to the free domain between them by shifting in the appropriate direction. The two access transistors at the extrema of the nanowire constitute the shift ports, which are used to inject a current pulse for moving the domains, thereby shifting the bits along the nanowire. Note that, a few extra domains on either end are reserved in order to ensure that there is no data loss as a result of shift operations.

3.2.1 Logical view of a multi-bit DWM cell

Fig. 3.3 shows a logical view of the multi-bit DWM cell with the ferromagnetic nanowire represented as a tape storing multiple bits and the read-write port shown as a tape head. In order to access a bit stored in the tape, we need to shift the head to

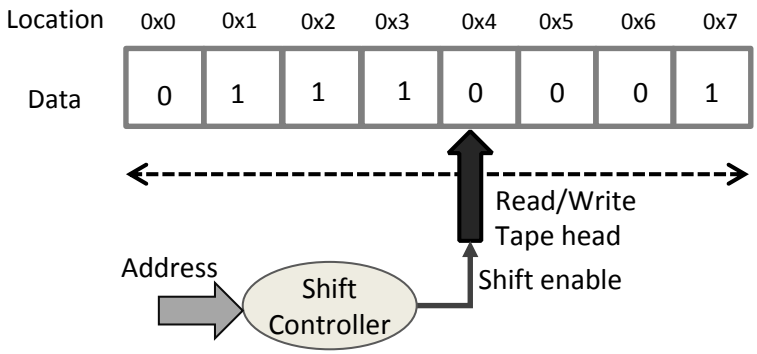


Fig. 3.3.: Logical view of a multi-bit DWM cell

the desired location on the tape via the shift controller and then perform the required operation. Therefore, the access latency for a given bit stored in such a structure is variable and depends on the number of shift operations performed. In fact, packing a larger number of bits in the tape leads to an increase in the number of shifts, resulting in higher access latencies. On the other hand, storing a smaller number of bits reduces the access latency at the cost of reduced cache capacity. DWM, therefore, presents an interesting trade-off between latency and capacity.

4. APPROXIMATE MEMORY COMPRESSION

4.1 Introduction

Modern computing systems, from servers to mobile devices, expend significant time and energy in moving data between processors and memory. Designers have explored larger on-chip caches, faster off-chip interconnects, 3D integration, and a host of other circuit and architectural innovations to address the processor-memory gap [78–81]. Yet, data-intensive workloads such as search, data analytics, and machine learning continue to pose increasing demands on off-chip memory systems, creating the need for new techniques to improve their energy efficiency and performance [1].

Many emerging applications that drive the demand for memory also exhibit intrinsic resilience, *i.e.*, an ability to tolerate approximations in the underlying computations or data while producing outputs of acceptable quality [9]. Approximate computing exploits this intrinsic resilience to improve energy and performance through techniques at various levels of the computing system stack, including software, architecture, and circuits [10, 11, 82, 83]. Adopting this approach to improve memory system energy efficiency, we propose *approximate memory compression* wherein data is transparently compressed (by introducing approximations within the memory controller) when it is written to off-chip memory and decompressed when it is read back on-chip. The reduced off-chip memory traffic leads to improvements in energy and performance, whereas the approximations made during compression potentially impact application output quality. We propose a set of hardware and software enhancements to realize this concept and demonstrate its utility across prevalent and emerging main memory technologies (DDR3 DRAM, LPDDR3 DRAM and STT-MRAM). To illustrate the feasibility of the proposal, we also design an FPGA prototype system

and perform measurements on the prototype system that reaffirm the benefits of approximate memory compression.

The majority of previous efforts in approximate computing focus on approximating logic while keeping memory accesses accurate. Efforts that have explored approximations in the memory subsystem [56, 64–73] differ from ours in their scope or the level of the memory hierarchy they target. Some of these efforts [64–68] focus on approximations within on-chip caches (*e.g.*, by merging cache lines with similar values or by returning approximate values upon cache misses). These techniques do not utilize any approximations when it eventually becomes necessary to access off-chip memory, which is the focus of this work. Other efforts improve the energy efficiency of emerging non-volatile memories [56, 84–86] using techniques such as reduced read/write durations or voltages. A final group of efforts [69–73] focus solely on reducing idle/refresh energy in DRAMs. Our proposal of approximate compression is complementary to these approaches, because it focuses on reducing the memory traffic and, therefore, can be used transparently in conjunction with most of them. Since our hardware changes are restricted to the memory controller, the proposed concept is easily applicable across a wide range of system architectures from general-purpose processors to accelerators and SoCs. While accurate or lossless memory compression has been explored in general-purpose processors [87–91] and embedded systems [92–94], we go beyond these efforts by rethinking memory compression from the perspective of approximate computing.

To enable approximate compression, we enhance the memory controller to transparently perform compression and decompression of data in a quality-aware manner while writing to/reading from off-chip memory. A key challenge in approximate computing is to perform approximations in a controlled manner so as to maintain application-level quality. Towards this end, we introduce a software interface that allows programmers to identify data structures that are amenable to approximations, thereby exposing specific memory regions to the proposed memory controller for approximate compression. We develop a runtime framework that automatically

determines numerical error constraints for the identified memory regions in order to maintain a desired application quality. Finally, the memory controller uses a fine-grained, dynamic precision scaling mechanism that exploits the varying opportunities present in different memory regions while adhering to their error constraints.

In summary, the key contributions of our work are:

- We propose approximate compression in main memories to improve energy and performance.
- We suggest a hardware design to realize approximate memory compression, in which a quality-aware memory controller transparently compresses and decompresses selected approximate memory regions while adhering to specified error constraints.
- To enable applications to utilize approximate compression, we propose an application programming interface to expose approximation-tolerant memory regions to the memory controller and a runtime framework to dynamically modulate the error constraints for each region.
- We demonstrate the applicability of the proposed approach in the context of an x86-based general-purpose processor system integrated with three different main memory technologies, *i.e.*, DDR3, LPDDR3, and DDR3 STT-MRAM. Our experiments on a wide range of machine learning benchmarks achieve significant benefits in memory energy (1.18× for DDR3, 1.52× for LPDDR3, and 2.0× for STT-MRAM based DDR3) and execution time (5.2% for DDR3, 5.4% for LPDDR3, and 9.3% for STT-MRAM based DDR3) for a negligible loss (0.3%) in application-level quality.
- To illustrate the feasibility of the proposed concepts in hardware, we also realize a prototype using the Intel UniPHY-DDR3 memory controller, Intel Nios II processor, and a Hynix DDR3 DRAM module on a Stratix-IV FGPA development board. Our measurements on 8 specific machine learning benchmarks reveal a

1.28× improvement in DRAM energy and a 11.5% improvement in execution time for ∼0.7% loss in application quality.

The rest of the chapter is organized as follows. Section 5.2 motivates the need for approximate memory compression. Section 4.3 describes the proposed approximate memory compression and the suitable hardware and software enhancements. Section 6.3 details the experimental methodology, and the results are presented in Section 5.7. Section 5.8 concludes the paper.

4.2 Motivation and Background

Approximate compression reduces off-chip memory traffic and is applicable to various memory technologies. For our exposition, we consider three different main memory technologies, *viz.*, DDR3 DRAM, LPDDR3 DRAM, and STT-MRAM based DDR3 memory.

Main memory organization. DRAM is the state-of-the-art memory technology for main memory design. A DRAM subsystem has a hierarchical structure, comprising of one or more *dual-inline memory modules* (DIMMs), each of which is in turn composed of several DRAM *chips*. In order to increase memory parallelism, DRAM chips may be grouped to form a *rank*. Each chip consists of multiple *banks* that can be accessed in parallel. A DRAM bank comprises of several *sub-arrays* with each sub-array being organized as a two-dimensional array of DRAM bit-cells, as shown in Fig. 4.1. Each sub-array also includes an array of *row buffers* that can hold data equivalent to an entire row of the sub-array.

Fig. 4.1 (inset) shows a DRAM bit-cell that consists of an access transistor and a capacitor. The logic state of the cell is determined by the charge stored in the capacitor, *i.e.*, fully-charged state represents "1" and discharged state represents "0". A read (or write) operation on the DRAM cell is performed by *pre-charging* the bitline (BL) to $V_{DD}/2$ and subsequently *activating* the corresponding wordline (WL). This results in a charge flow from the capacitor to the bitline (or vice versa). The charge

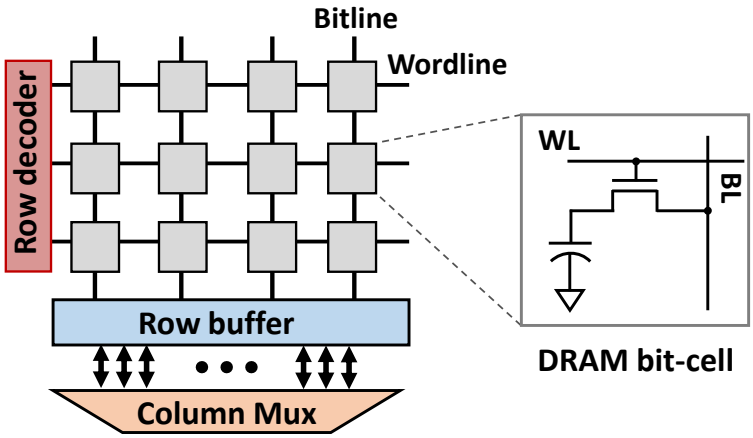


Fig. 4.1.: DRAM sub-array

sharing between the capacitor and the bitline causes for a change in the bitline voltage which is then sensed by a sense amplifier to drive the bitline with logic "0" or "1" depending on the logic state of the cell.

To maintain data integrity, DRAM systems require periodic refresh operations, which consume significant energy. Spin-Transfer Torque Magnetic RAM (STT-MRAM), has recently emerged as an alternative non-volatile memory technology for main memory design [3, 95]. In contrast to DRAMs, STT-MRAMs are non-volatile, *i.e.*, the data is retained even when no power is supplied. As a result, STT-MRAMs do not require refresh operations periodically, eliminating the refresh energy overheads in main memory. While prior efforts on approximate memory have targeted reducing refresh energy in DRAM, we focus on reducing the access (read/write) energy making it applicable across all memory technologies.

Motivation. Fig. 4.2 motivates approximate memory compression by quantifying the potential for reductions in memory traffic across a suite of machine learning benchmarks (workload details are provided in Section 6.3). These benchmarks use multi-dimensional arrays of native data types (*i.e.*, `char`, `short`, `int`, *etc.*). The baseline implementations were already optimized to use the smallest data types possible without impacting application quality. Next, we identified data structures amenable to approximation, and for each such data structure, determined a maximum permis-

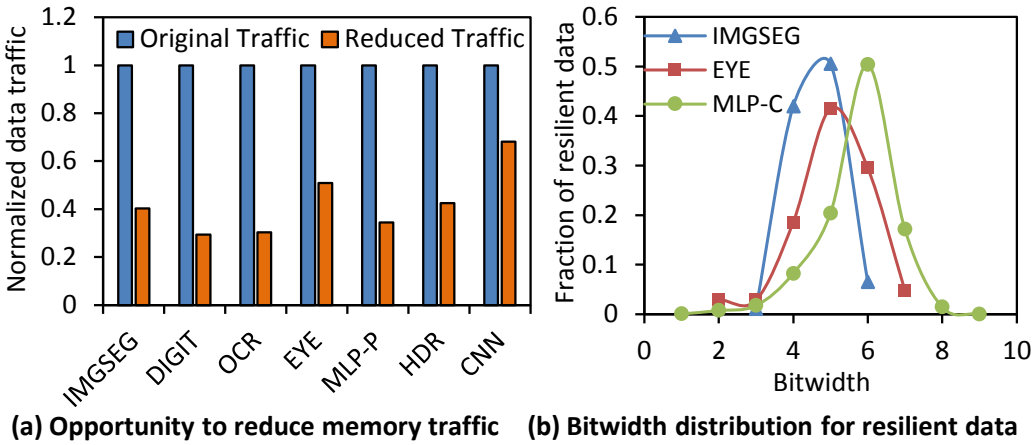


Fig. 4.2.: Motivation for approximate memory compression

sible error magnitude for its elements such that application quality was maintained within 1% of the baseline. Using techniques described in this paper, the precision of each 32-byte memory block was reduced to the minimum that satisfied the error magnitude. With this approach, we observed (Fig. 4.2(a)) that it is possible to lower memory traffic by up to 3.5× (2.5× on average). Fig. 4.2(b) presents, for three representative benchmarks, the distribution of bitwidths to which elements in various memory blocks can be compressed.

These results underscore the significant opportunity available for reducing memory traffic by approximating data structures in a fine-grained manner, well below the precision granularities supported in software. However, the key challenges in tapping this potential are: (i) how do we support fine-grained approximation in hardware?, (ii) how will the reduced precision translate to reduced off-chip memory traffic?, and (iii) how can applications benefit from approximate memory compression without significant increase in programming complexity? We address these challenges by enhancing the memory controller to transparently perform fine-grained, dynamic precision scaling and packing/unpacking of memory blocks, and by proposing a software abstraction and runtime framework to minimize programmer effort.

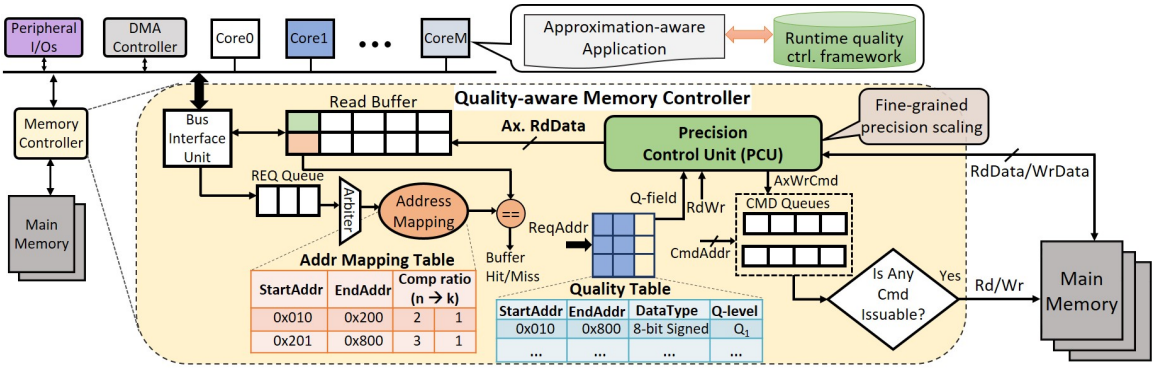


Fig. 4.3.: Approximate memory compression: Overview

4.3 Approximate Memory Compression

Fig. 4.3 provides an overview of a system with approximate memory compression, with the accompanying hardware and software enhancements. A quality-aware memory controller interfaces to a standard memory (*e.g.*, DRAM) and provides the ability to compress and decompress different memory regions to different accuracy levels. The software enhancements consist of an interface to specify approximation-tolerant memory regions and a runtime quality control framework that regulates accuracy constraints on each of these memory regions to meet a given application-level quality. In this section, we discuss the approximate compression scheme followed by a description of the hardware and software enhancements.

4.3.1 Approximate Compression Scheme

Consider a block to be written to memory that comprises a set of fixed-size scalar elements (*e.g.*, 8-bit, 16-bit or 32-bit). Let us also suppose that an accuracy constraint is provided for this block that specifies the maximum error that may be incurred in each element during compression. For example, Fig. 4.4 shows an 8-byte uncompressed memory block that consists of eight unsigned 1-byte integers, and the maximum error magnitude that may be incurred in compressing each element is 2. The

proposed approximate compression scheme analyzes the values in the uncompressed memory block to determine two key parameters — the number of MSB bits (M) and LSB bits (L) that may be truncated while satisfying the specified accuracy constraint. Such a bi-directional precision scaling scheme exploits both the lower numerical significance of LSBs and the higher likelihood of MSBs being zero or sign-extended[1]. Since most applications demonstrate significant heterogeneity in the values of M and L across memory blocks, we determine the values of M and L at runtime in the memory controller on a per-block basis, and embed them as a *header* in the compressed memory block itself. In addition, we also store two padding bits (P_L and P_M) that are used to fill in the LSB and MSB values during decompression. In the example of Fig. 4.4, the 1-byte elements can be compressed to 4-bits each, with an additional byte to store the header (M, L, P_M, P_L). Consequently, the entire block can be compressed to 5 bytes, resulting in a savings of 3 bytes. In practice, memory accesses cannot be performed in fractional block sizes; therefore, we compress n blocks to k blocks where n and k are integers[2] and $n > k$.

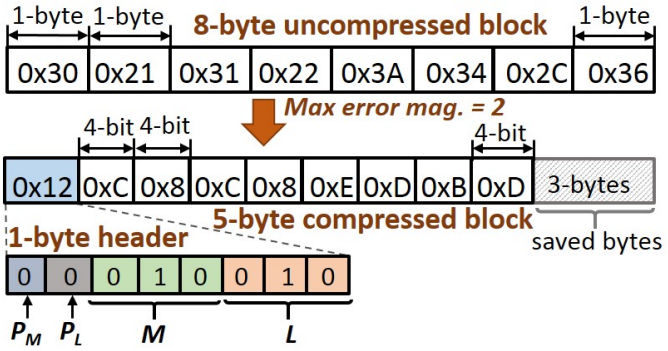


Fig. 4.4.: Approximate compression scheme

To decompress a compressed block, we use the header to determine the (M, L, P_M, P_L) parameters used in compression. Accordingly, the compressed data elements

[1]For signed data values, we exclude the sign bit from being truncated.

[2]We found that $n \leq 8$ and $k \leq 2$ captures most of the opportunity while limiting the complexity of the compression and decompression logic.

are padded with L LSBs of value P_L and M MSBs of value P_M (excluding the sign bit for signed data). Fig. 4.5 shows the decompression of the compressed block from Fig. 4.4. In this case, 2 LSBs and MSBs for each element are set to "0" to obtain the uncompressed block.

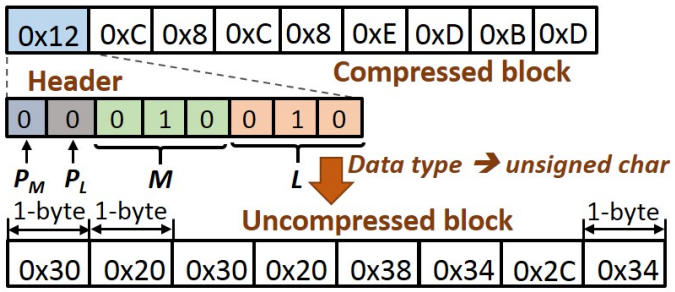


Fig. 4.5.: Approximate decompression scheme

4.3.2 Quality-aware Memory Controller

Fig. 4.3 presents the design of a quality-aware memory controller that realizes the approximate compression scheme described above. The controller is enhanced with: (i) A *Quality table* that specifies approximate memory regions and their associated error constraints, (ii) A *Precision control unit* that performs the compression and decompression, and (iii) An *Address mapping table* that converts the addresses of uncompressed memory blocks[3] to the addresses of compressed blocks in main memory. In the following paragraphs we discuss these components in detail.

Quality table. In order to perform compression in a quality-constrained manner, the memory controller requires a mechanism to identify the memory blocks that are amenable to approximation. We propose a small, fully-associative quality table for this purpose. As shown in Fig. 4.3, each entry in the quality table includes: (i) an *approximate memory region*, or a memory address range that stores approximation-

[3]The processor and other hardware components that are behind the memory controller are oblivious to the compression.

resilient data (ii) the data type associated with the specified address range, *e.g.*, signed 8-bit, unsigned 16-bit *etc.*, and (iii) the error constraint (*e.g.*, maximum error magnitude) for memory locations within the range. On each write, the memory block address is compared with the address ranges present in the quality table. In case of a match, the corresponding error constraint and data type are used to compress the memory block. If no matching entry exists, the block is written to memory as is. For a read request, in case of a match, the quality table indicates the data type of the elements in the compressed block. The data type is then used to appropriately decompress the block read from memory. Since the quality table has a very small number of entries (8 in our implementation), it does not introduce significant area, power, or performance overheads.

Precision control unit. The precision control unit (PCU) performs compression and decompression of memory blocks using the proposed scheme. The PCU takes the data type and error constraint from the quality table, and a read/write control signal ($RdWr$) as inputs.

During a write operation, the PCU determines the number of MSB bits (M) and LSB bits (L) that can be dropped for each element in the uncompressed block without violating the error constraint. It also determines the padding values (P_L and P_M) to be used for the LSB and MSB bits. The PCU includes a small write buffer (8 entries in our implementation) that holds memory blocks that need to be written to main memory. The PCU packs multiple consecutive blocks into a smaller number of compressed blocks depending on the compression ratio achieved by the chosen (M, L) and, subsequently, issues writes to main memory.

During reads, the PCU uses the compression parameters (M, L, P_M, P_L) from the header within the compressed block, as well as the data type of the elements in the block (from the quality table). The compressed block is decompressed using the technique described in Section 5.5.2. The resulting decompressed blocks are stored in a small fully-associative read buffer (8 entries in our implementation). During subsequent read operations, the incoming address is compared with all the read buffer

entries in parallel for a hit/miss. If this results in a buffer hit, the matching uncompressed block is directly read from the buffer, thereby avoiding a main memory access.

Address mapping table. A key challenge with the proposed compression scheme is that it modifies the addresses at which blocks are stored in memory. This is further complicated by the fact that different compression ratios may be used for different blocks based on their error constraints and data values. We enhance the address generation logic already present in memory controllers with an address mapping mechanism that seamlessly translates an uncompressed main memory address to the corresponding compressed location in main memory. Specifically, we introduce an address mapping table as shown in Fig. 4.3, in which each entry consists of: (i) the start and end address of a contiguous uniformly compressed region (CUCR) in which memory blocks are compressed with the same ratio, and (ii) the compression ratio for the CUCR.

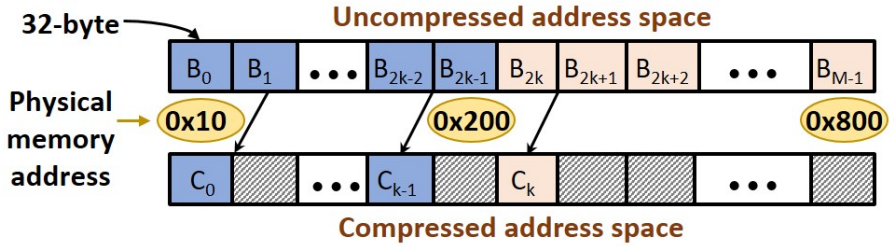


Fig. 4.6.: Address mapping in the proposed design

Fig. 4.6 illustrates the proposed address mapping scheme. Blocks B_0, B_1, ..., B_{2k-1} form a CUCR with a compression ratio of 2, while blocks B_{2k}, B_{2k+1}, ..., B_{M-1} have a compression ratio of 3. When compressed block C_0 (corresponding to B_0 and B_1) is written to memory, an entry is created in the address mapping table with the address of the first and the last uncompressed block (0x10 and 0x11) and a compression ratio of 2 (*i.e.*, $n = 2$ and $k = 1$). If successive addresses are compressed with the same ratio, the address mapping table entry is simply extended (in the

example, this happens until location 0x200 for B_{2k-1}). However, if the address is not contiguous with an existing range in the table, or if the compression ratio is different (in the example, block C_k corresponding to uncompressed blocks B_{2k}, B_{2k+1}, B_{2k+2} has a compression ratio of 3), a new entry is allocated in the table. Fig. 4.3 lists the address mapping table entries at the end of compression for all the blocks illustrated in this example. Note that the start address of each CUCR is the same as its physical main memory address. For any block within the CUCR, its offset from the start address is simply scaled by the compression ratio. Therefore, compression results in unused locations in the compressed address space (shaded in gray) at the end of each CUCR[4].

During memory reads, the address mapping table entries are compared with the incoming read address to obtain the CUCR address range and the compression ratio. The CUCR start address and the compression ratio are used along with the input read address to compute the physical main memory address.

4.3.3 Software Support for Approximate Memory Compression

In order to minimize the programmer effort required to utilize approximate memory compression, we propose a simple interface that can be used to identify data structures that are resilient to approximation, and a runtime quality control framework (QCF) that automatically determines the data structure-level error constraints so as to maintain the target application quality.

Identifying approximate memory regions. Most workloads contain a mix of resilient data structures that can be approximated and sensitive data structures that cannot tolerate approximations. Therefore, it is essential to identify and selectively approximate the former. Adopting a similar philosophy to many previous efforts in approximate computing, we introduce a function `set_approximate_memory` that the programmer can call (as shown in Fig. 4.7) to indicate a memory region that can be

[4]Our objective is to reduce memory traffic rather than overall memory footprint. This allows us to greatly simplify the address translation between compressed/uncompressed blocks.

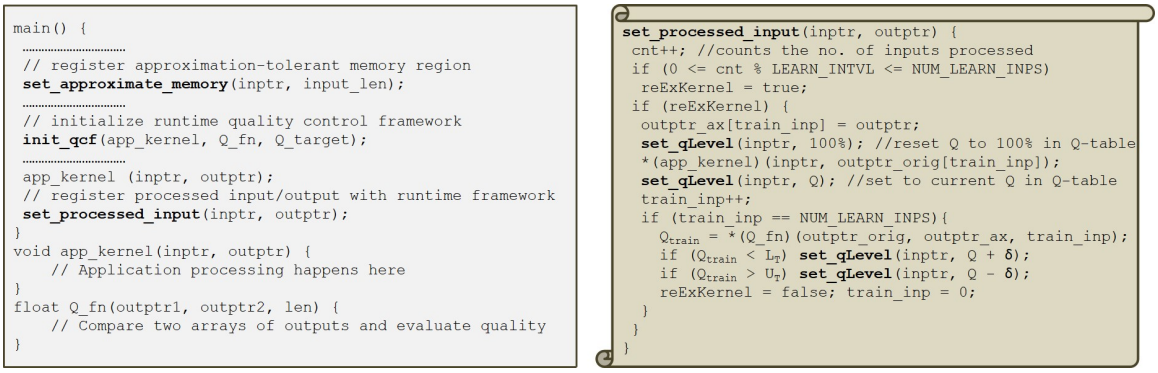


Fig. 4.7.: Modified application with the runtime quality control framework (QCF) for approximate memory compression

approximated. This function creates an entry in the quality table, which is exposed to software through memory-mapped registers in the quality-aware memory controller.

Runtime Quality Control Framework. Fig. 4.7 shows the structure of an application modified to utilize the proposed QCF. The programmer initializes the QCF by calling function `init_qcf` to register the application kernel (`app_kernel`), a quality function (`Q_fn`) written by the programmer that evaluates quality by comparing two arrays of outputs produced by `app_kernel`, and the target application-level quality constraint (`Q_target`). In addition, as discussed above, the programmer identifies memory regions that can be approximated by calling `set_approximate_memory`. The application also reports each input processed by the application kernel to the QCF by calling the `set_processed_input` function, details of which are shown in the right half of Fig. 4.7.

The QCF modulates the error constraints for the approximate memory regions by interleaving two phases: (i) an *exploration* phase, in which it dynamically learns new values of the error constraints for each memory region, and (ii) an *exploitation* phase, where the system operates with the learned error constraints to improve memory system efficiency. Fig. 4.8 summarizes the operation of these phases over time at a high-level. The exploration phase, realized within the `set_processed_input` function as shown in Fig. 4.7, is periodically invoked at a specified learning in-

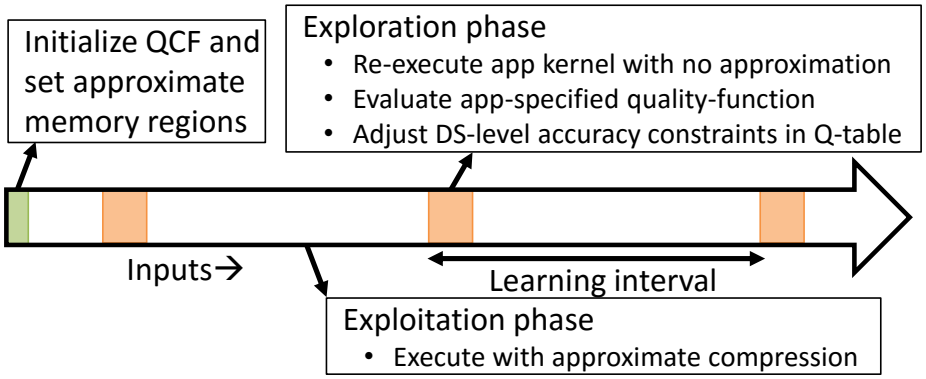


Fig. 4.8.: Timeline for different phases in QCF

terval (`LEARN_INTVL`). For each input processed during this phase, the QCF re-executes `app_kernel` without any approximation (*i.e.*, setting the error constraints to 0 for all approximate memory regions), and records the output produced with and without approximation. At the end of the exploration phase (when `train_inp = NUM_LEARN_INP`), `Q_fn` is called to compare the recorded outputs and evaluate application-level quality. The evaluated quality (Q_{train}) is compared against two thresholds (L_T and U_T) derived from the specified target quality (`Q_target`) to determine whether quality needs to be improved or relaxed. Specifically, the error constraints are reduced or tightened by δ, the smallest granularity at which precision scaling is beneficial for approximation. The QCF ranks approximate memory regions based on their access counts (tracked by the memory controller) and employs a greedy approach to select a memory region and tighten or loosen its error constraints. To avoid oscillatory behavior of reduction and increase in error constraint for a region, the QCF caps how often the constraint for each approximate memory region may be updated.

4.4 Experimental Methodology

In this section, we describe the experimental setup and application benchmarks used to evaluate the proposed approximate memory compression technique.

4.4.1 Experimental setup

To evaluate the proposed technique, we simulate a general-purpose processor system integrated with different main memory technologies. We also designed an FPGA prototype system to demonstrate the feasibility of the proposed concepts in hardware.

Table 4.1.: System configuration

Processor Core	x86, in-order processor, 2 GHz
L1 I/D-cache	32KB/32KB, direct-mapped, 32B line size
L2 unified cache	256KB, 8 way-set associative, 32B line size
Cache latency	L1: 2-cycle, L2: 6-cycle
Memory Cntrl.	close-page policy, FR-FCFS with queuing model, 16-entries RD and WR queues
Memory Params	1 channel, 64-bit I/O **DDR:** DDR3-1600, 4Gb, 1KB page-size **LPDDR:** LPDDR3-1600, 4Gb, 1KB page-size **STT-MRAM:** DDR3-1600, 256Mb, 512 bit page-size

General-purpose processor system. We model the proposed quality-aware memory controller in NVMain [96], which is then integrated as a model for the main memory system in the gem5 architectural simulator [97]. Table 5.2 shows the system configuration used in the evaluation. The DDR3 and LPDDR3 timing and power parameters for the evaluation were obtained from Micron‘s datasheets [98, 99]. We also estimate the benefits of our proposal with STT-MRAM, an emerging non-volatile memory technology for main memory design, using the timing and power characteristics of the recently offered 256Mb DDR3 Spin-Torque MRAM module from Everspin [100]. The energy and performance overheads of the proposed hardware enhancements were also reflected in our experiments.

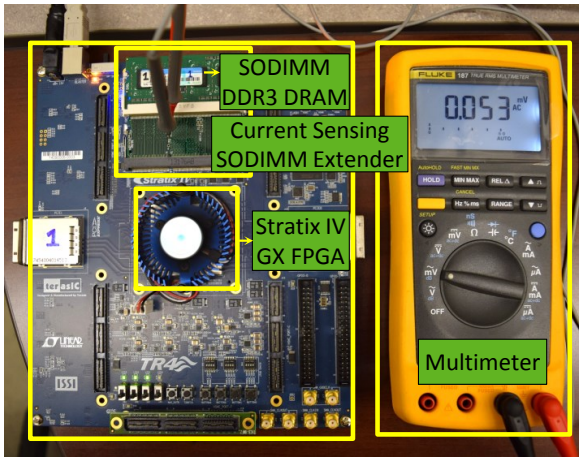


Fig. 4.9.: FPGA prototype system for the proposed scheme

FPGA prototype. Fig. 4.9 shows the prototype system using a Stratix-IV GX FPGA development board. It consists of a Nios-II processor [101] running at 266.67 MHz that interfaces with a Hynix 1GB DDR3 DRAM module through a high-performance UniPHY DDR3 DRAM controller [102]. The off-chip memory access latency in our system was 100 processor cycles. We modified the UniPHY DRAM controller to implement the proposed approximate memory compression technique. The area and power overheads for the proposed enhancements in the memory controller were found to be ∼3.9% and ∼4.3%, respectively. To measure the power consumed by the DRAM, we used an SODIMM extender with an in-built current sensing resistor and measured the voltage drop across the resistor using a high precision multimeter. We utilized a performance counter in the Nios-II processor to measure application execution time.

4.4.2 Benchmark applications

Table 5.3 lists the benchmarks, the associated algorithm, and the datasets used in our experiments along with the metric used to evaluate output quality in each case.

Table 4.2.: Application benchmarks for approximate memory compression

Application	Algorithm	Dataset	Quality metric
Text Classification (Text-CL)	Support vector machines	REUTERS	Classification accuracy (percentage of inputs correctly classified)
Handwritten Digit Recognition (HDR)		MNIST	
Digit Classification (LeNet-5)	Convolutional neural networks (LeNet-5 architecture)		
Character Recognition (OCR)	K-nearest neighbors	OCR digits	
Handwritten Digit Classification (CLAS)		Gisette	
Eye Detection (EYE)	Generalized learning vector quantization	YUV faces	
Object Recognition (MLP-C)	Multi-layer perceptron	CIFAR-10	
Protein Structure Classification (MLP-P)		Protein	
Image Segmentation (IMGSEG)	K-means clustering	Berkley dataset	Mean cluster radius
Optical Character Clustering (DIGIT)		OCR digits	
Document Clustering (REAL)		Real-sim	

4.5 Experimental Results

This section presents the results of various experiments that demonstrate the benefits of the proposed approximate compression scheme across a range of system architectures.

4.5.1 System performance benefits

Fig. 4.10 summarizes the improvements in application execution time achieved using the proposed quality-aware memory controller in the x86 processor-based system across different main memory technologies. In our evaluation, we consider three different baseline main memories: (i) a DDR3 DRAM system, (ii) an LPDDR3 DRAM, and (iii) an STT-MRAM DDR3 main memory. The execution time for each benchmark is normalized to the corresponding baseline design with no memory compression.

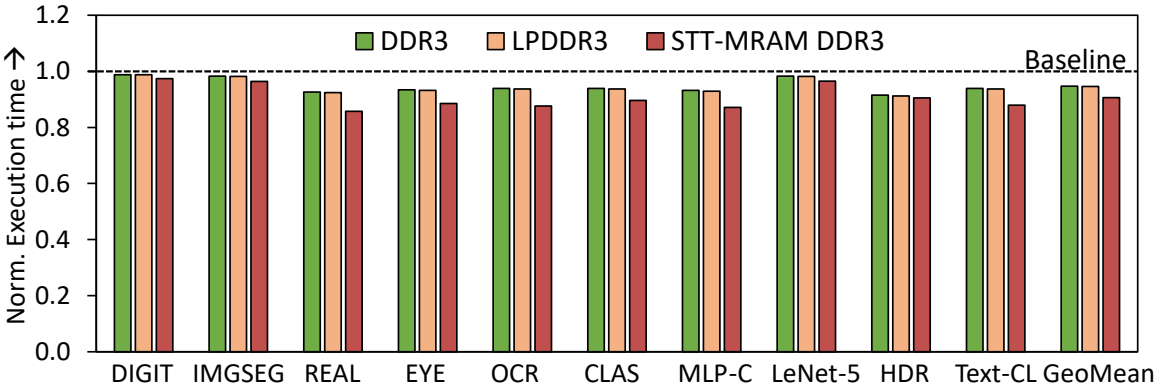


Fig. 4.10.: System performance improvements across different memory technologies with approximate memory compression

Across all benchmarks, approximate memory compression lowers the execution time by up to 8.5% (5.2% on average) for a negligible loss (average 0.3%) in application-level quality compared to the baseline DDR3-based system. For the LPDDR3 and the STT-MRAM DDR3 system, the average performance benefits further increase to 5.4% and 9.3%, respectively. The benefits in performance are primarily due to the reduction in main memory traffic, resulting in less data being written to or read from main memory. Note that, the extent of performance improvement for each application depends on its memory latency sensitivity and also the total memory intensity. For instance, the benefits in performance for a subset of compute-intensive benchmarks (DIGIT, IMGSEG and LeNet-5) are much lower than the other benchmarks across different memory technologies. The benefits for the STT-MRAM DDR3 system is higher since the memory latency is higher in case of STT-MRAMs.

4.5.2 Memory energy improvements

Fig. 4.11 illustrates the main memory energy savings obtained using approximate memory compression over the three baseline memory systems. For the DDR3 system, we achieve energy benefits ranging from 1.06× to 1.34× (average of 1.18×) for an average 0.3% loss in output quality. The energy benefits further extend to 1.16×–

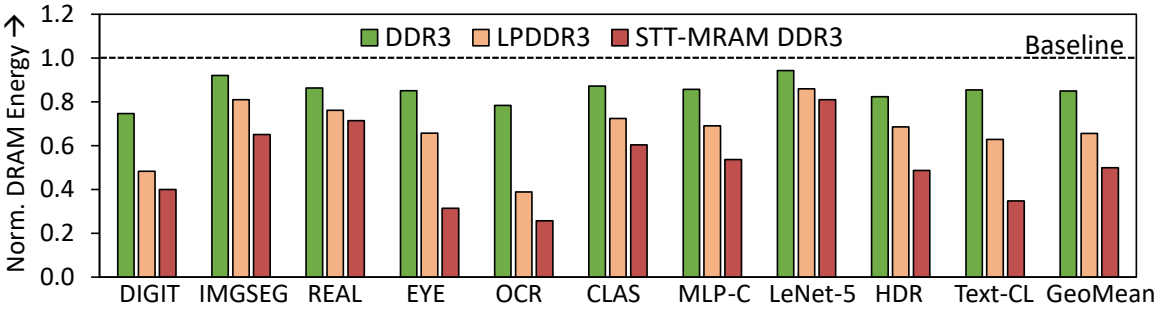


Fig. 4.11.: Main memory energy benefits across various memory technologies with approximate memory compression

2.57× (average of 1.52×) and 1.24×–3.88× (average of 2.0×), for the LPDDR3 and STT-MRAM systems, respectively.

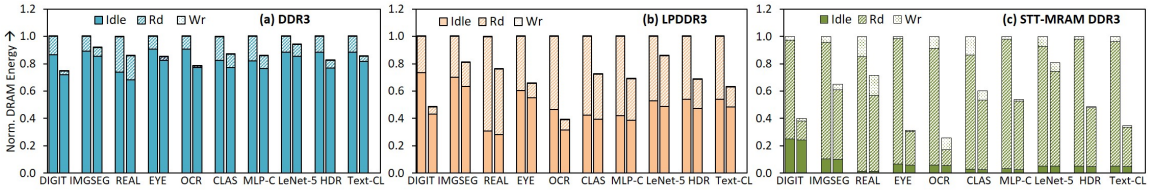


Fig. 4.12.: Main memory energy breakdown across different memory technologies with approximate memory compression

We next present a breakdown of different energy components, *viz.*, read, write and idle energy[5], that contribute to the main memory energy. Fig. 4.12 shows the main memory energy breakdown for each benchmark across the three memory technologies, *i.e.*, DDR3, LPDDR3 and STT-MRAM DDR3, with and without memory compression. For the baseline DDR3 system, read, write and idle energies respectively constitute 13.8%, 0.1%, and 86.1% of the total DRAM energy consumption. In this case, the idle energy is substantially higher primarily due to the high background/steady-state power consumption of the DDR3 DRAM module. In contrast, the LPDDR3 and the STT-MRAM based main memories that consume much less standby power com-

[5]The idle energy comprises of both refresh energy and the background peripheral circuit energy.

pared to DDR3 DRAM, have the read, write and idle energy fractions as 47.3%, 0.1%, 52.6% and 86.9%, 6.1%, 7.1%, respectively, across all benchmarks. Note that, the idle energy fraction in STT-MRAM based DDR3 is even lower due to the non-volatile nature of DRAM cells, resulting in the refresh energy being completely eliminated for the STT-MRAM based DRAM module. On the other hand, the write energy fraction in STT-MRAM DDR3 is much higher because of the high write energy in STT-MRAMs. On average, we achieve 2.5× and 1.7×, reduction in read and write energy, across all memory technologies. The read and write energy benefits are predominantly due to fewer read and write accesses to main memory with approximate memory compression. The average idle energy benefits vary across memory technologies, with 1.07× for the STT-MRAM based DDR3 system, 1.1× for the standard DDR3 memory system, and 1.18× for the LPDDR3 system. The improvements in idle energy are largely dependent on the reduction in execution time. However, it also depends on the fraction of the runtime spent in the memory state when any bank is open (active-idle) and when all the banks are closed (precharge-idle). This is because DDR3 and LPDDR3 memories have considerable difference in their active-idle and precharge-idle power with the active-idle power being significantly higher than the precharge-idle power. Hence, for a subset of benchmarks, *i.e.*, DIGIT, OCR, LeNet-5 and HDR, the benefits in idle energy extend even beyond the reduction in execution time due to a reduced number of main memory accesses arising from the proposed memory compression approach. Consequently, a higher fraction of the total runtime is spent in the precharge-idle state, which in turn lowers the total idle energy. Fig. 4.13 presents a breakdown of the execution time spent in different memory states (active-idle, precharge-idle, and RD/WR), for one such application, corresponding to both the cases, *i.e.*, with no memory compression and when subject to approximate memory compression. As shown in the figure, the ratio of runtime spent in active-idle and precharge-idle states dramatically changes from 3.3 to 0.3 as as result of the proposed memory compression, leading to significant benefits in idle energy.

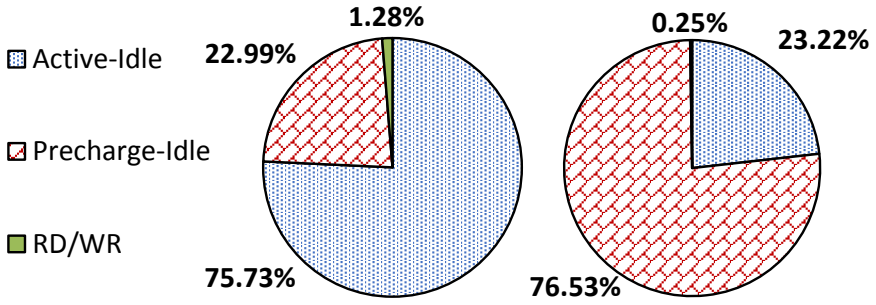


Fig. 4.13.: Runtime breakdown for the DIGIT application executing on a DDR3 DRAM system

4.5.3 FPGA prototype system results

We next present the energy and performance results measured on the FPGA prototype system designed to demonstrate the feasibility of the proposed concepts.

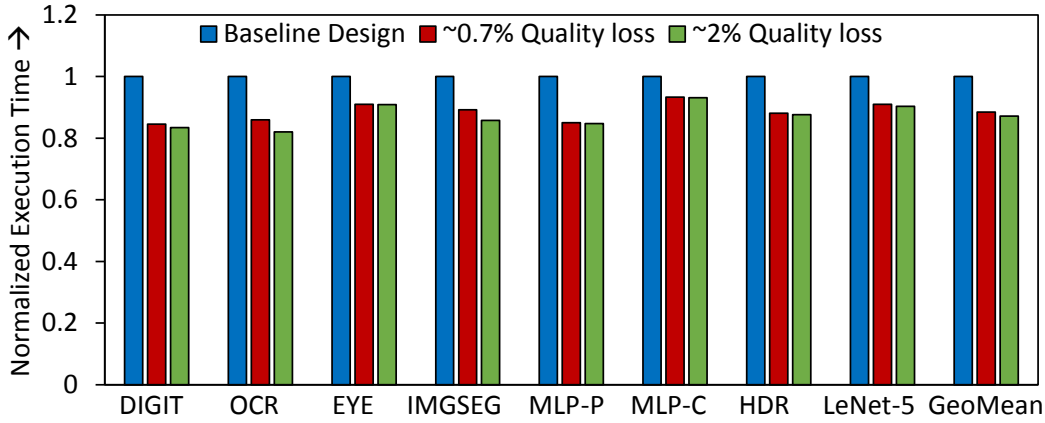


Fig. 4.14.: System performance improvement in the FPGA prototype with approximate memory compression

Execution time benefits. Fig. 4.14 shows the normalized execution time of the benchmarks for different application-level output quality constraints, using the proposed design. The execution times are normalized to a baseline design with a memory controller that does not perform memory compression. As shown in the figure, approximate memory compression results in a reduction of up to 15.4% (average of

11.5%) in execution time for a small degradation in output quality (average 0.7%). Relaxing the quality target to an average 2% degradation, results in a reduction of up to 17.9% (average of 12.8%) in execution time. The execution time benefits are higher than the x86 processor-based system discussed above since the NIOS-II core only supports an L1 data and instruction cache in contrast to the x86 processor with two levels of cache hierarchy, leading to increased memory latency sensitivity of applications in this case. Note that the reduction in execution time directly translates to a reduction in full-system energy consumption.

DRAM Energy Benefits. Fig. 4.15 plots the normalized DRAM energy consumption for the two application-level quality targets. The energy values are normalized to the baseline design described above. As shown, approximate memory compression results in energy improvements of 1.23× to 1.38× (average of 1.28×) for an average 0.7% loss in output quality. For a more relaxed quality target (2% average degradation in output quality), we observe energy benefits of up to 1.4× (average of 1.32×).

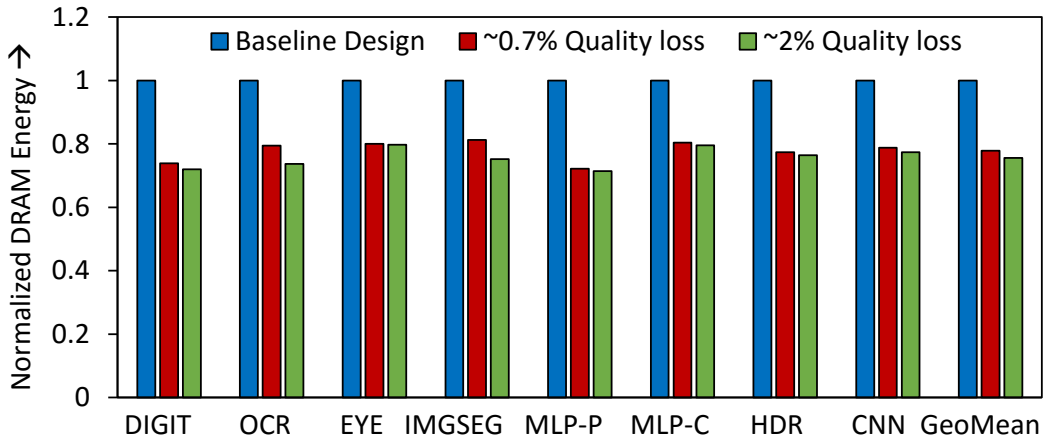


Fig. 4.15.: DRAM energy savings in the proposed FPGA system

DRAM Energy Breakdown. To provide further insight, we also show a breakdown of the DRAM energy consumption into read, write, and idle energy. Fig. 4.16 illustrates this breakdown for our benchmarks at the two quality levels. We observe that read, write, and idle contribute 16%, 0.2%, and 83.8% respectively, to the total DRAM energy in our baseline design. The average read and write energy benefits from the

proposed approximate compression scheme are 5.03× and 1.85×, respectively, for ∼0.7% degradation in output quality. The read energy reductions are higher since the benefits of approximate memory compression are amplified by the read buffer, which enables decompressed data to be reused. Writes do not see this benefit, since in our benchmarks writes exhibit much lower locality than reads. The idle energy reduction is between 1.1× and 1.2×, which is directly proportional to the reduction in execution time. Note that, the idle energy measurements in our case does not fully reflect the fine-grained transition between the different memory states as discussed in Section 4.5.2. For a more relaxed quality specification (∼2%), we obtain even higher compression ratios, that translates to average benefits of 7.11×, 2.62×, and 1.15× in read, write, and idle energy, respectively.

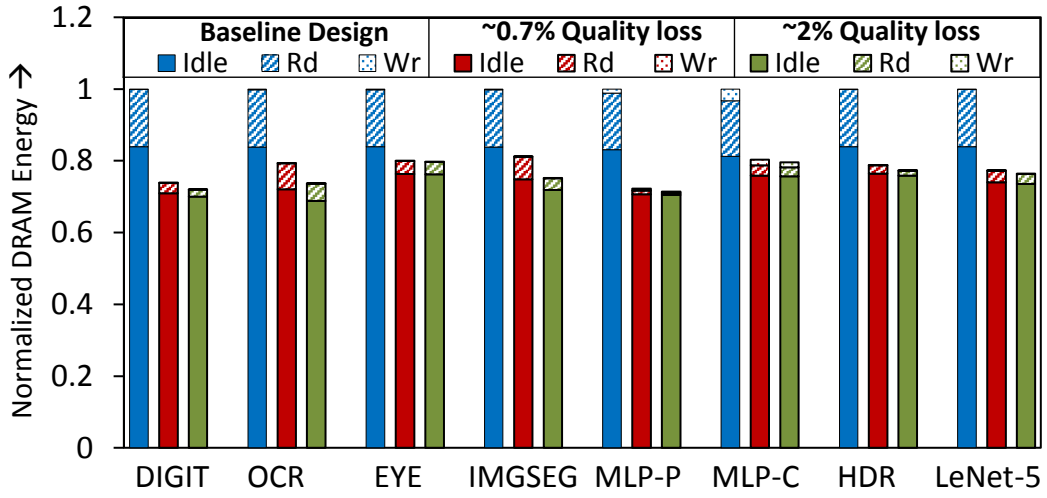


Fig. 4.16.: DRAM energy breakdown in the proposed FPGA system

4.5.4 Comparison with a uniform approximation scheme

Next, we demonstrate the effectiveness of using fine-grained precision scaling for compression at runtime, as proposed in this work, by comparing it with a simpler design-time approach where all the memory blocks are uniformly approximated using precision scaling. Fig. 4.17 shows the normalized memory traffic and the loss

in application-level quality for different data structure (DS) level error bounds for two different applications. As seen in the figure, we obtain a superior memory traffic and application-level quality when compared to uniform approximation, clearly demonstrating the benefits of the proposed approach.

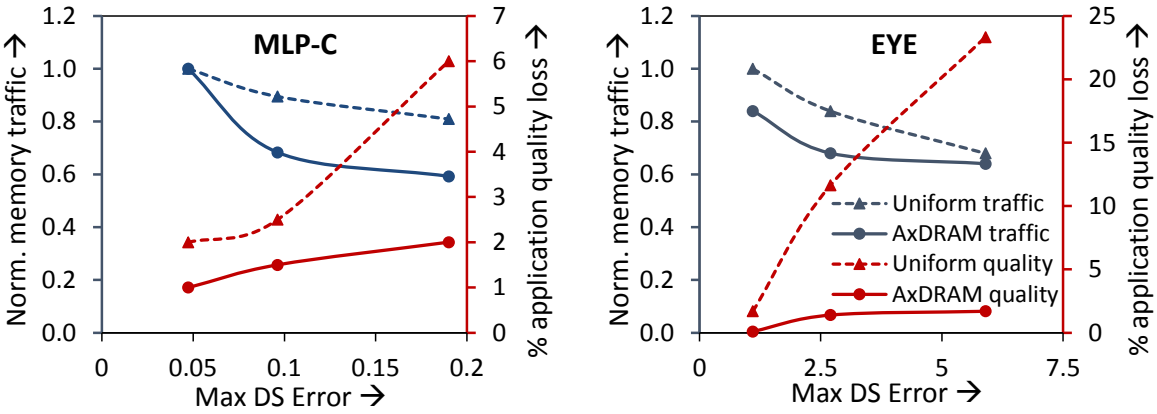


Fig. 4.17.: Memory traffic benefits over uniform approximation

4.5.5 Comparison with a lossless compression scheme

Fig. 4.18 compares the normalized memory traffic observed using the proposed scheme with a lossless bi-directional compression scheme discussed in Section 5.5.2, wherein the error constraints were set to 0. Across all benchmarks, we obtain a `2.35`× reduction in memory traffic compared to the lossless scheme for an average `0.6%` loss in application-level quality, highlighting the ability of the proposed compression scheme to achieve even higher compression ratios by exploiting the error-resilience of the applications.

4.5.6 QCF in action: Case study

Fig. 4.19 shows a timeline with the proposed QCF in action for the OCR application, wherein the DS-level error bound is self-tuned at runtime so as to maintain the target application-level output quality. It also shows the evaluated quality for the

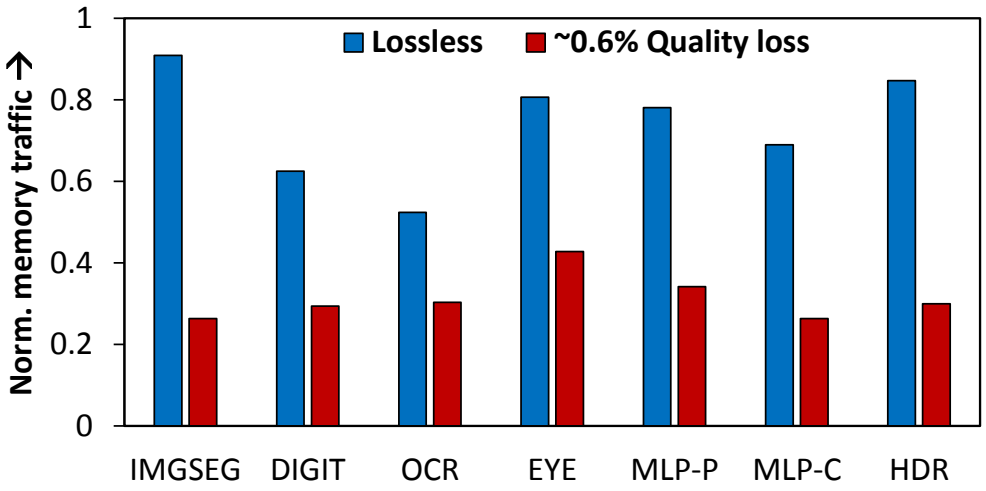


Fig. 4.18.: Memory traffic reduction over a lossless compression scheme

inputs processed during each exploration phase and the upper and lower threshold used to determine the appropriate action, *i.e.*, relax or tighten the DS-level error constraint. In this case, we deliberately vary the application-level quality target from 95% to 85% (t = 110 in Fig. 4.19). The runtime framework is able to successfully adapt to this change by suitably modulating the DS-level error constraints. It also illustrates the ability of the framework to recover from an undershoot in quality (t = 130 in Fig. 4.19).

4.6 Summary

This chapter discussed *approximate memory compression* that exploits the error resilience of applications to reduce memory traffic, and thereby improve energy and performance. We designed a quality-aware memory controller that transparently realizes approximate memory compression, making it applicable across a wide range of computing systems. We utilized a dynamic, fine-grained precision scaling mechanism to achieve high compression while providing control over output quality. We proposed a programmer interface and a runtime framework to enable applications to use approximate memory compression. Our experiments on a diverse range of main

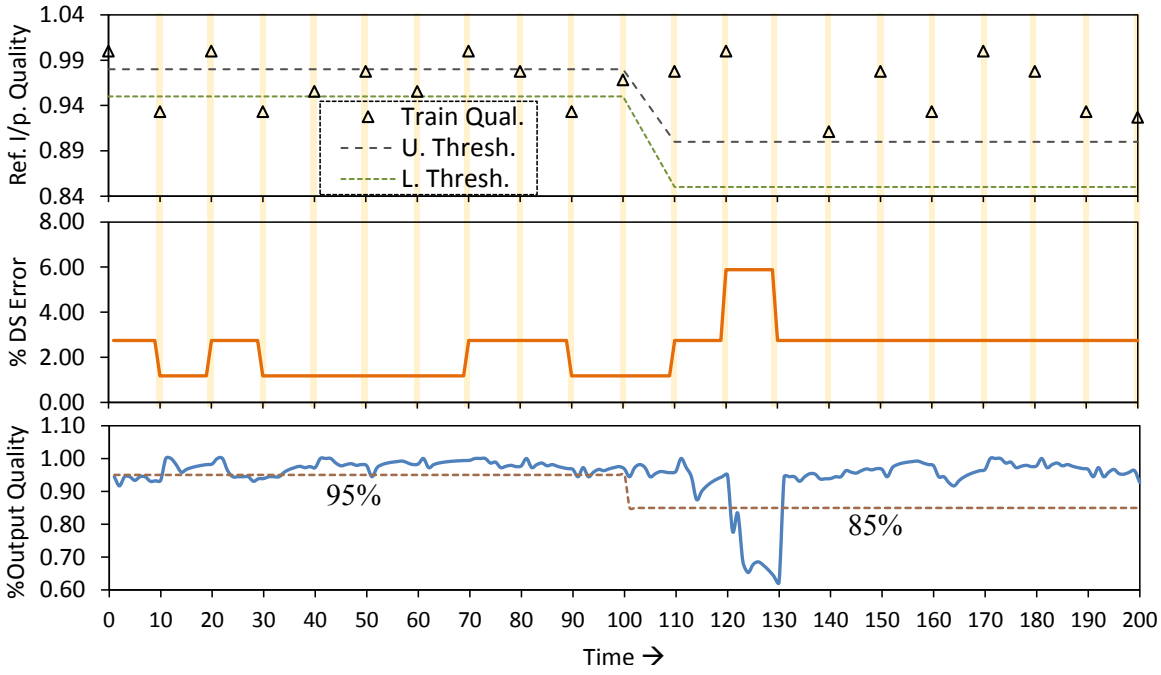


Fig. 4.19.: Output quality *vs.* time for the OCR benchmark

memory systems along with measurements from an FPGA prototype suggest that approximate memory compression achieves substantial improvements in performance and energy.

5. APPROXIMATE MEMORY DESIGN FOR ENERGY-EFFICIENT SPINTRONIC MEMORIES

5.1 Introduction

Spin transfer torque magnetic RAM (STT-MRAM) has gained significant interest in recent years as a potential post-CMOS memory technology [3, 4, 103]. STT-MRAMs offer high density and near-zero leakage, making them promising candidates for on-chip memories. Although STT-MRAMs greatly reduce leakage energy compared to CMOS memories, their overall energy efficiency is still limited by the energy required for spin-transfer torque (STT) switching in writes and reliable single-ended sensing during reads. Several promising research efforts have been devoted to improving the energy efficiency of STT-MRAM at the device, circuit and architecture levels [31, 32, 36–41, 103]. Representative techniques at the architecture level include utilizing a hybrid STT-MRAM-CMOS memory hierarchy that directs frequently written blocks to the CMOS region [36,37], and eliminating redundant writes by tracking and updating only changed bits [38,39]. While we share the objective of these efforts, we propose the use of a different approach – approximate storage – to achieve energy efficiency for STT-MRAMs.

Several emerging applications that have fueled the demand for larger on-chip memories (multimedia, recognition, data mining, search, and machine learning, among others) also exhibit intrinsic resilience to errors, *i.e.*, the ability to produce results of acceptable quality even with approximations to their computations or data [9]. Approximate computing exploits this characteristic of applications to derive energy or performance benefits using techniques at the software, architecture, and circuit levels [10, 11, 104, 105]. Most previous work in approximate computing focuses on processing or logic circuits. Previous efforts on approximate storage [56, 65, 67–69, 74]

can be classified based on the level of the memory hierarchy that they target. Some focus on secondary storage and main memory [56, 69, 106] using techniques that are complementary to our work. Others focus on application-specific memory designs [74]. A few efforts [65, 67, 68] explore approximate cache architecture with CMOS memories, using techniques such as skipping cache loads on misses. Complementary to these efforts, we explore new approximate memory design techniques that exploit the specific error-energy trade-offs that manifest in STT-MRAMs to improve energy efficiency of both read and write operations. Further, we propose a quality-configurable memory array, that allows reads and writes to be performed at varying quality levels based on application requirements. We also illustrate its utility as a scratchpad in the memory hierarchy of a domain-specific vector processor and as an L2 cache in the context of a general purpose processor through suitable architectural and software enhancements.

A key challenge in approximate computing is how to manage the approximations so as to obtain the most favorable energy *vs.* application quality trade-off. To this end, we explore a combination of various circuit- and architecture-level techniques that yield significant energy benefits for small probabilities of read, write and retention errors. We explore: (i) *Approximation through partial reads/writes*, where reads (or writes) are to selected least significant bits are ignored, (ii) *Approximation through lower read currents*, wherein a lower read current is used for sensing, thereby trading off decision failures for read energy benefits, (iii) *Approximation through skipped writes*, wherein writes to a cache block are skipped at run-time if they are similar to its current contents, (iv) *Approximations through incomplete writes*, wherein the write current, write duration or both are lowered, resulting in an increased probability of write failures, and (v) *Approximations through skipped refreshes*, wherein refresh operations to the low retention blocks are selectively skipped.

Next, employing the above mechanisms, we design a *quality-configurable* memory array (QcMem), in which read and write operations to each word in the memory can be performed at various predefined levels of quality at runtime. Towards this end, we

enhance QcMem with low-overhead circuits to dynamically modulate the read/write current (and duration), and ignore read/write to each data bit.

To evaluate the benefits of QcMem, we suggest two different on-chip memory designs. First, we propose STAxPad (Spintronic Approximate scratchpad), a scratchpad in the memory hierarchy of a domain-specific vector processor [105]. To expose STAxPad to software, we enhance the load/store instructions in the ISA of the programmable vector processor with quality fields that denote the error tolerable during their execution. We also develop an auto-tuning framework that utilizes gradient descent search to determine the quality fields of the load/store instructions in a given application program so as to minimize energy for a desired application output quality. Finally, we propose STAxCache (Spintronic Approximate Cache), an STT-MRAM based approximate L2 cache for general purpose processors that preserves the full flexibility of a conventional cache, while enabling different levels of approximation to different parts of a program's memory address space. STAxCache utilizes a heterogeneous cache organization that is composed of low retention and high retention cache ways, each of which is in turn designed using QcMem. We enhance the cache replacement policy to exploit the heterogeneous cache ways for achieving lower write energy. To provide control over the errors introduced during approximations, the STAxCache architecture also consists of: (i) a *quality table* that captures the quality requirements for the regions of address space and also tracks the number of refreshes skipped for each region, (ii) a *quality-aware cache controller* that enforces the specified quality constraints. We introduce a software interface that enables programmers to specify quality requirements with minimal effort, and ISA extensions that allow these requirements to be conveyed to the underlying cache hardware.

In summary, the key contributions of this work are:

- We explore the use of approximate storage in STT-MRAMs to improve their energy efficiency. We identify and characterize mechanisms at the circuit- and architecture- level that enable disproportionate energy benefits at the cost of small probabilities of read, write or retention errors.

- We utilize the above techniques to realize a quality-configurable memory (QcMem) array, in which data can be stored at varying levels of accuracy based on the application requirements. We demonstrate the utility of the proposed QcMem array as a scratchpad in the context of a programmable vector processor and as an L2 cache in the context of general purpose processors.

- We introduce extensions in the ISA and propose various mechanisms that expose the QcMem array to software.

- We develop a device-to-architecture simulation framework to evaluate our proposal. Our experiments on a wide range of machine learning benchmarks reveal 1.67× improvement in scratchpad energy and 1.44× improvement in cache energy over an iso-capacity STT-MRAM based on-chip memory for $< 0.5\%$ loss in output quality.

The rest of the chapter is organized as follows. Section 5.2 motivates the need for approximate storage in STT-MRAMs. Section 5.3 outlines the approximation techniques explored and describes the design of QcMem. Section 5.4 details the proposed ISA extensions and the auto-tuning framework for STAxPad. Section 5.5 describes the STAxCache architecture, the ISA enhancements and the software support required to expose it to programmer. Section 6.3 details the experimental methodology, and the results are presented in Section 5.7. Section 5.8 concludes the chapter.

5.2 Case for Quality-configurable memories

Figure 5.1 compares the different energy components of an iso-capacity cache (1MB) designed using SRAM and STT-MRAM [8, 39]. The energy components are normalized to the corresponding energy components in SRAM. As shown in the figure, STT-MRAM greatly lowers the leakage energy compared to SRAM owing to the non-volatile nature of STT-MRAM. However, the benefits in leakage energy do not extend to the read and the write energy, thereby limiting the overall energy-efficiency

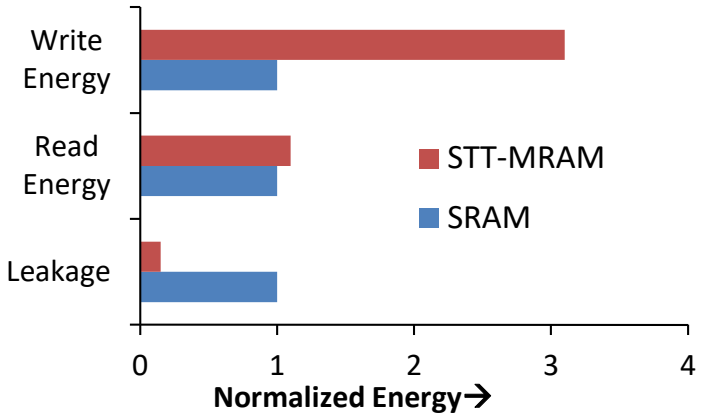


Fig. 5.1.: Energy comparison SRAM *vs.* STT-MRAM

of STT-MRAM. To address the read and write energy inefficiencies, we propose to use approximate storage that trade-offs energy at the cost of approximations in read/write operations. However, not all read/write operations can be approximated, therefore, it is important to have control over the errors introduced during memory operations. We next motivate the need for controlled approximations during read/write operations through two representative benchmarks – eye detection and optical character recognition. These benchmarks were compiled to the ISA of a domain-specific vector processor [105]. We employ an error injection framework similar to [9] in order to identify the memory instructions amenable to approximation and introduce errors in the read/written data. The errors introduced in each instruction were random in nature yet bounded to be within a predetermined maximum error magnitude constraint. Subsequently, for different application-level quality targets, the error magnitude constraint was varied to obtain the number of memory instructions in the benchmark that can be approximated at runtime.

Figure 5.2 illustrates the fraction of vectored load/store dynamic instructions in the ISA that can be executed in an approximate manner *vs.* the loss in application output quality for different instruction-level error magnitude constraints. As shown in the figure, we observe a significant increase in the fraction of instructions that can be executed approximately with bounded instruction-level error magnitude for any given

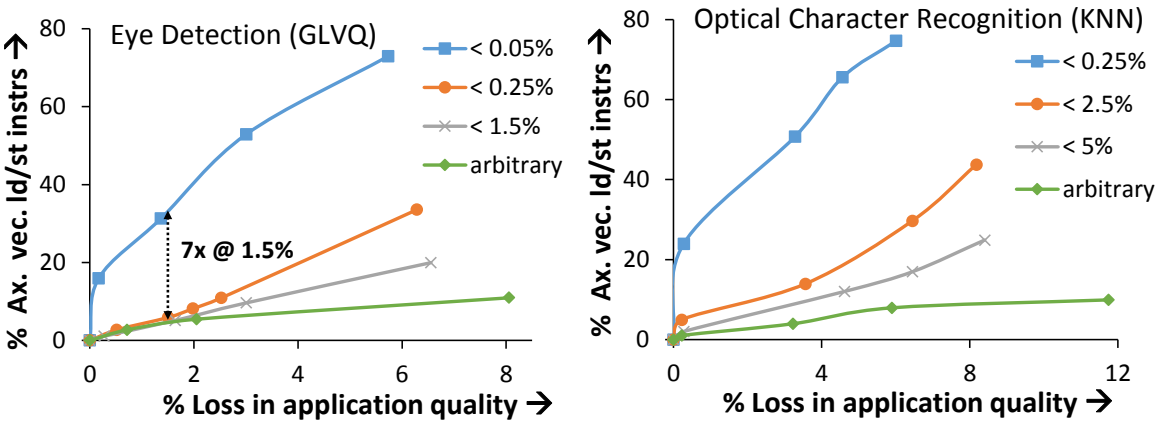


Fig. 5.2.: Fraction of instructions subject to controlled approximations for two different applications.

application output quality. Specifically, for the eye detection benchmark, consider the case when the loss in application quality is 1.5%. In this case, for an error magnitude constraint of $< 0.05\%$ at the instruction-level, the fraction of instructions that can be approximated increases by 7× compared to the case when errors of arbitrary magnitude are introduced. For this benchmark, with an instruction-level bound of $< 0.05\%$, across different output quality targets, we observe an increase of 7×–10× in the percentage of load/store instructions that can be executed approximately. Similarly, for the optical character recognition benchmark case, we observe an increase of 10×–18× in the fraction of memory instructions that can be subject to approximations for an error bound of $< 0.25\%$ of the maximum value when compared to the arbitrary error scenario. These results underscore the significance of introducing approximations in a controlled manner for maximizing the opportunity to approximate memory instructions, which in turn reduces the overall memory access energy.

A key challenge for approximate storage in a cache is that data that can tolerate varying levels of approximation (or no approximation at all) can be loaded into the same cache line at different times. Figure 5.3 shows the fraction of error resilient cache blocks that are observed over time for an image segmentation application when executing on an x86 processor with a 2MB L2 cache. In this case, we utilize two levels

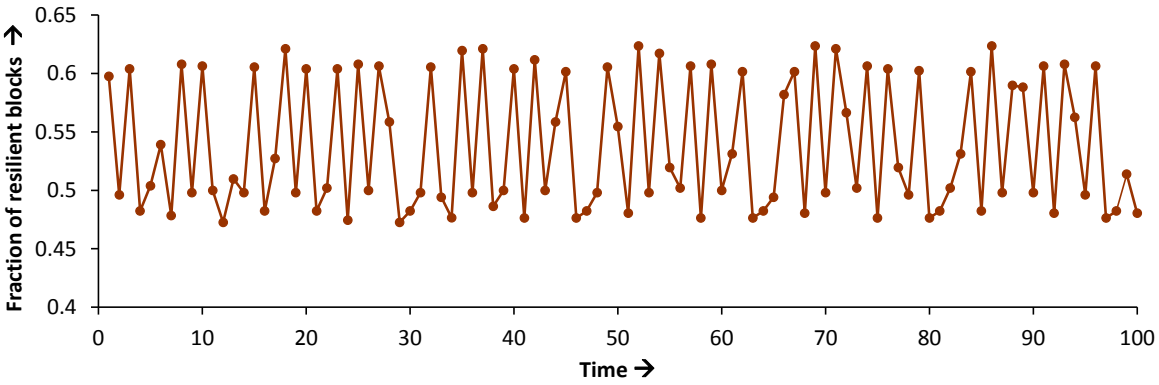


Fig. 5.3.: Variation in the fraction of resilient blocks in a cache over time with image segmentation application.

of approximation, *i.e.*, accurate and approximate, and dynamically capture the cache utilization of the application during execution using a cycle-accurate simulator. We observe a significant variation in the number of error-resilient blocks present in cache at different times with the fraction of blocks varying from 47% to 62% in a given time duration (*e.g.* between $t = 10$ and $t = 20$). Therefore, we require hardware-based mechanisms that provide control over the approximations at runtime for such a cache while still retaining the full flexibility of a conventional cache.

In this chapter, we address these challenges by designing a quality-configurable spintronic memory that has the ability to modulate the extent to which read/write operations are approximated at runtime, and by proposing mechanisms in hardware and software to provide control over the approximations such that a beneficial energy *vs.* quality trade-off is obtained at the application level.

5.3 Quality configurable spintronic memory

In quality configurable memories, each read and write operation can be performed at different predefined levels of accuracy (and energy) based on the requirements of the application. To enable quality configurable memory design in the context of spintronic memories, we explore different circuit-level techniques that yield a favorable

energy *vs.* quality trade-off. These mechanisms are then utilized at the array-level to realize quality configurable read/write operations. In this section, we describe the approximation mechanisms and the array-level design in detail.

5.3.1 Approximation techniques

To obtain energy efficient reads and writes in STT-MRAM, we explore five different approximation mechanisms that are discussed in detail in the following paragraphs.

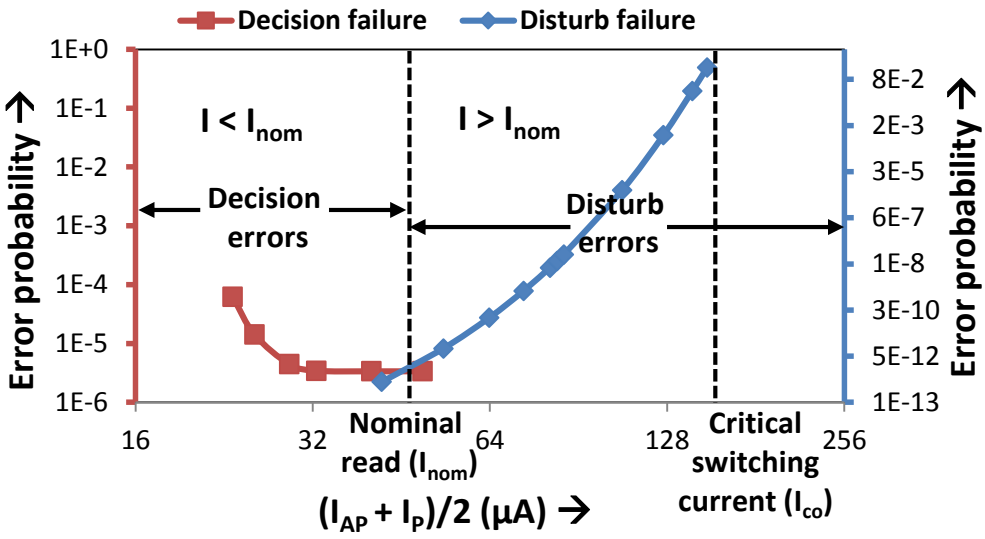


Fig. 5.4.: Read decision and disturb trade-off for an STT-MRAM bit-cell

Approximations through incorrect read decisions. The logic state stored in the bit-cell is determined by measuring the effective resistance of the cell (MTJ + access transistor). This is achieved by driving the voltage across BL and SL to a predefined nominal value (V_{read}) and utilizing a sense amplifier to compare the current flowing through the bit-cell with a reference current. Now, if V_{read} is decreased, the difference in magnitude of current through the MTJ when a 0 *vs.* 1 is stored also decreases. Consequently, with a reduced sensing margin, process variations in the MTJ or in the access transistor have an increased chance of leading to incorrect values at the

sense amplifier output, resulting in an increased probability of read errors, as shown in Figure 5.4. Further, decreasing V_{read} improves the energy consumption of the read operation. Quantifying this trade-off, Figure 5.5(a) shows the energy *vs.* decision error probability for an STT-MRAM bit-cell obtained by performing Monte-Carlo simulations under process variations (σ/μ for t_{ox} = 2% and MTJ area = 5%) [107] based on the modeling framework described in Section 6.3. We find that modulating V_{read} yields considerable benefit in energy, *e.g.*, over 2X improvement for an error probability of $\sim 10^{-5}$.

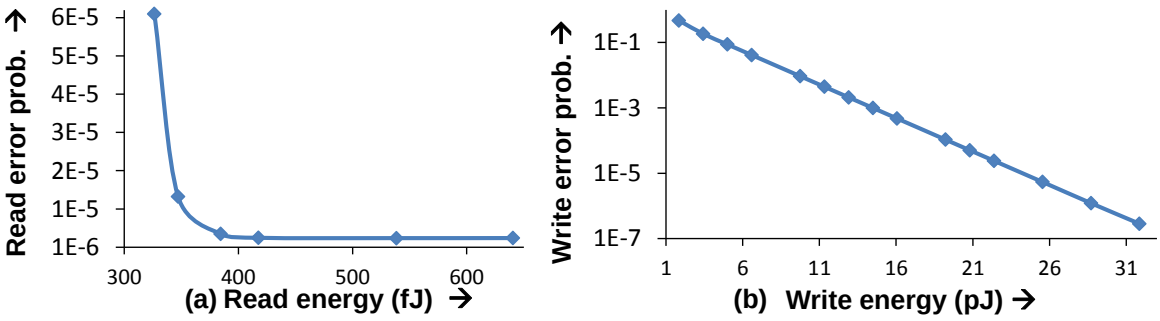


Fig. 5.5.: Energy *vs.* error probability trade-off for an STT-MRAM bit-cell

Approximations through read disturbs. In contrast to the previous technique, in this case, V_{read} is increased. This allows for the duration of read to be decreased, resulting in improved performance. However, as shown in Figure 5.4, when the read current (I_{read}) nears the critical switching current (I_{co}) of the MTJ, the value stored in the bit-cell gets erroneously flipped with a probability ($P_{disturb}$) given by Equation 5.1 [4].

$$P_{disturb} = 1 - \exp\left(-t_{read}/\tau_0 \exp\left(\Delta\left(1 - I_{read}/I_{co}\right)\right)\right) \tag{5.1}$$

In Equation 5.1, τ_0 refers to the attempt period ($\sim$ 1ns) and Δ is the thermal stability factor of the MTJ. Although the above technique enables faster reads, we note that read operations are seldom a performance bottleneck for STT-MRAM memories, hence we typically operate in a regime of lower V_{read}, which results in lower energy consumption.

Approximations through incomplete writes. The third mechanism focuses on approximate writes, where we leverage the stochastic nature of STT switching. For a successful write, a current higher than the critical switching current of the MTJ is injected for a specific duration. If either the write current or duration is lowered, then the stochastic nature of STT switching, captured analytically by the Sun model [108], results in writes to fail with a probability given in Equation 5.2.

$$P_{write} = \exp\left(-t_{wr}/\langle t_{sw}\rangle\right) \tag{5.2}$$

In the above equation, t_{wr} is the write pulse duration, $\langle t_{sw}\rangle$ is the average switching delay of the bit-cell which is inversely proportional to the write current [4]. Leveraging this insight, we propose lowering the write duration below $\langle t_{sw}\rangle$ in order to obtain approximate writes. Naturally, a lower write duration leads to improved write energy. Figure 5.5(b) quantifies this trade-off by plotting the write energy *vs.* error probability obtained through bit-cell simulations.

Approximations through skipped writes or partial reads/writes. This technique focuses on approximate reads as well as writes, wherein we either ignore an entire write operation or specific bits (particularly, a few least significant bits or LSBs) while reading/writing a memory word. Unlike SRAM, STT-MRAM does not suffer from the half-select problem; therefore, bit lines and source lines corresponding to the LSBs may be gated to achieve energy savings. During approximate reads, the gated LSBs are simply set to 0 in the value returned from the bit-cell. In contrast, for approximate writes, we first perform a read to compare the difference in magnitude of the incoming data with the previously stored value. Depending on the error constraint, we either choose to skip the entire write operation while retaining the stale value, or ignore write to LSBs, preserving the previously stored values to these bits, in turn saving considerable energy.

Approximations through lower retention time. Another key design metric of an STT-MRAM bit-cell is the retention time, which is the duration for which the data stored in an idle bit-cell is retained. Lowering the retention time reduces the write

energy required to switch the MTJ, since it decreases I_{co} for the MTJ. However, it also makes the bit-cell more prone to retention failures due to thermal disturbances.

5.3.2 Quality-configurable array design

In this section, we describe the quality configurable STT-MRAM based memory array (QcMem) that utilizes the approximation mechanisms described above. Figure 5.6 shows the array-level view of the proposed design. A QcMem is associated with an additional input (Q) that denotes the quality desired during each access. A quality decoder interprets the quality input and generates control signals (Q_{RD}/Q_{WR}) that regulate the overall quality of memory access. In order to regulate the quality, the memory is provisioned to either access the different bits in a word with varying error probabilities or skip the access to these bits to allow partial read/write as discussed above. Accordingly, as shown in Figure 5.6, the quality control signals for each column (i) of the memory are generated individually by the quality decoder logic ($Q_{RD}[i]/Q_{WR}[i]$) based on the quality input. Note that the quality control signal corresponding to each column is a multi-bit value *i.e.,* each bit in the memory can be either gated to enable partial read/write or accessed with multiple predefined error probabilities. Towards this end, the peripheral circuits present in each column are enhanced with low-overhead logic ($< 0.5\%$ of the total area) to gate the corresponding bitline and sourceline of the bit-cell or modulate the bit-cell read current (or write duration) as described below.

Peripherals for read quality configurability. Read peripherals in an STT-MRAM array typically include a read bias circuit to drive the read voltage to the desired value and a sense amplifier for comparing the resultant current. Figure 5.6 shows the transistor-level schematic of the read peripheral circuit enhanced to modulate the read current between two quality levels (*i.e.*, accurate and approximate) or skip the read. It consists of two current mirrors with differently sized transistors and a gating transistor; one of the current mirrors is sized for nominal read current

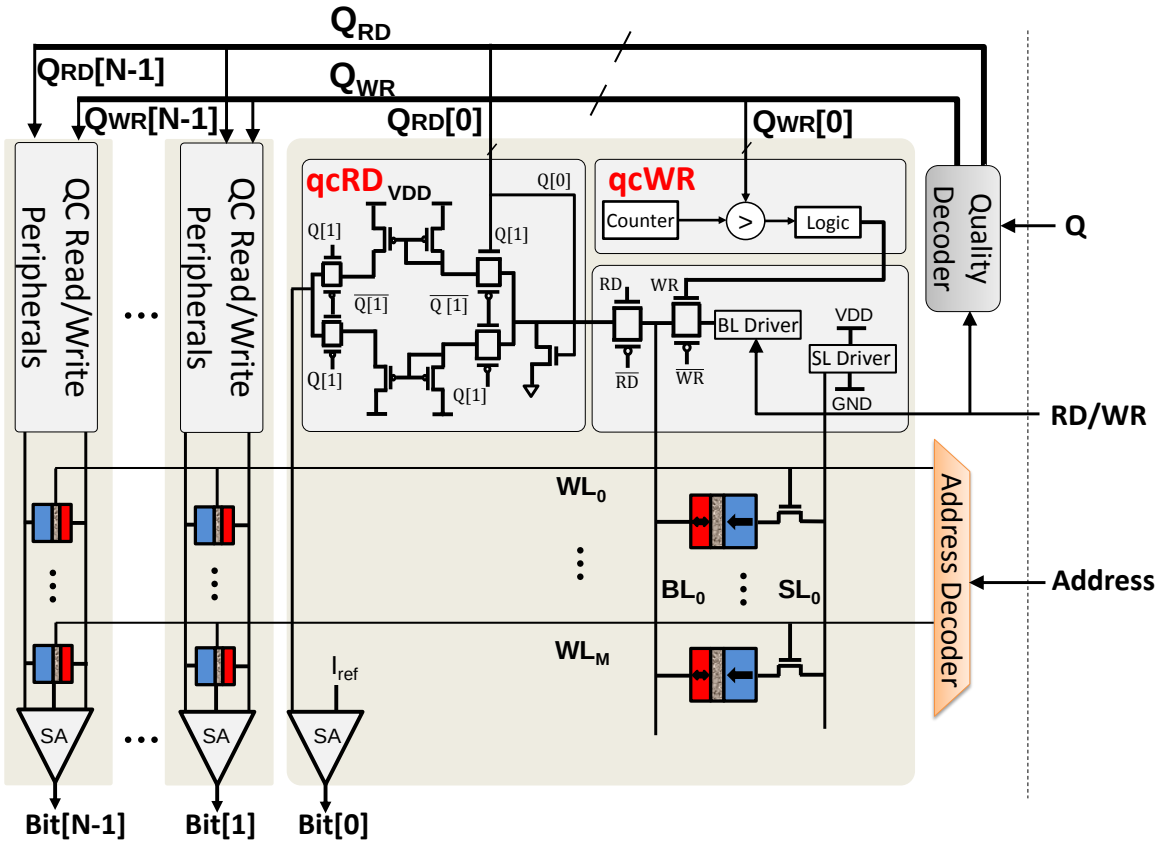


Fig. 5.6.: Quality-configurable memory array

flowing through the MTJ while the other is downsized for a substantially lower read current that results in an increased probability of read errors. The read peripheral circuit is also provisioned with a gating transistor that skips the read to the bit. The quality control signal (Q_{RD}) generated by the quality decoder dynamically selects between the current mirrors or the gating transistor, resulting in the bit lines being either clamped to different voltages or disabled altogether, during read operation. This scheme can be easily extended to support additional quality levels for each bit line by using additional current mirrors and enabling a subset of them based on the desired quality.

Peripherals for write quality configurability. In the case of writes, the number of cycles for which the write current is supplied to the bit-cell is modulated[1]. We achieve this by using a counter and a comparator circuit whose threshold for comparison (Q_{WR}) is determined by the quality decoder.

5.4 STAxPad: Scratchpad with QcMem

To evaluate the benefits of QcMem array at the application level, we integrate it as a scratchpad in the memory hierarchy of a vector processor described in [105]. Vector processors can efficiently execute applications from several important domains by exploiting their fine-grained data parallelism and regular data access patterns. The following subsections describe the vector processor architecture and the architectural enhancements required to utilize STAxPad.

5.4.1 Vector processor architecture

Figure 5.7 shows a block diagram of the processor architecture. It consists of a three tiered hierarchy of processing elements: (i) 2D-array processing elements (2d-PEs) arranged as a two dimensional array, (ii) 1D-array processing elements (1d-PEs) located along the borders of the 2D array, and (iii) a scalar processing element (scalar-PE).

The memory hierarchy of the vector processor is also shown in Figure 5.7. The processor contains the following micro-architectural registers: (i) 2 sets of streaming memory elements (SMs), which are First-in-First-out (FIFO) buffers of equal length, used for feeding input data operands to the 2D- and 1D-processing elements, (ii) One accumulator register in each 2d-PE that holds its result and can be scanned out to memory, (iii) An accumulator and a register file in each 1d-PE, and (iv) A general-purpose register file in the scalar PE. Besides these registers, the vector processor has an on-chip data and instruction memory used to load (store) data

[1]When the number of cycles is zero, write to the bit is ignored.

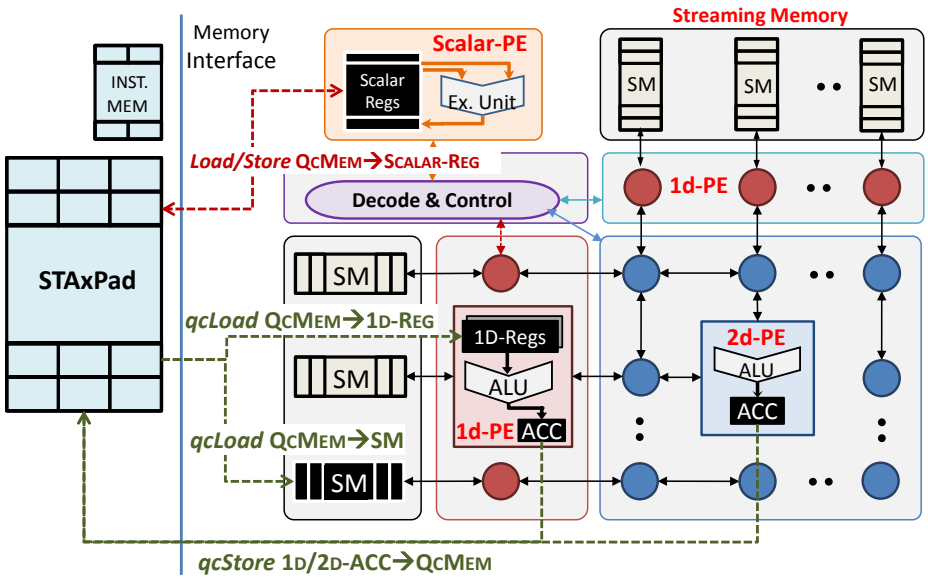


Fig. 5.7.: Vector processor architecture for STAxPad evaluation

and instructions to (from) the processing elements. Figure 5.7 shows the load/store instructions in the ISA of the vector processor. The ISA comprises of two classes of memory instructions, *viz.* scalar load/store instructions to/from the scalar-PE registers and vector load/store instructions to the SM, 1d- and 2d-PE registers. A key feature of the architecture is that it provides a natural separation of computations and data that are resilient to approximations from those that are not. The 1d- and 2d-PEs typically operate on data amenable to approximations, while the scalar PE is used to perform control operations such as loop control, pointer arithmetic *etc.* that need to be accurate. We leverage this property and approximate only the memory regions accessed by the vector load/store instructions. Furthermore, we allow software to specify which vector loads/stores may be approximated, and by how much.

5.4.2 Quality-aware load/store instructions

We introduce the notion of quality-aware memory instructions that expose the inherent resilience of the applications to the QcMem. Towards this end, we extend

the vector load/store instructions with *quality fields* that are used to express the required accuracy with which these memory instructions should be executed. We next discuss the proposed instruction format and the manner in which approximations are performed using these memory instructions.

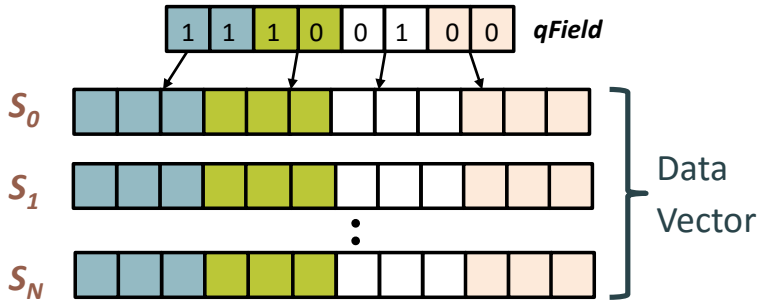


Fig. 5.8.: Example illustrating *qField* mapping to bit groups

Instruction format. We associate a quality field with every vector load/store instruction to indicate the maximum expected error rate[2] that can be tolerated during the corresponding read/write operations. For example, a vector load instruction is represented as:

$$\textbf{qcVecLoad}\ \ destReg,\ address,\ length,\ qField \tag{5.3}$$

The quality field ($qField$) is specified as a vector of expected error rate, wherein each element represents an expected error rate for a group of bits within the data word. The number of bit groups is an important design parameter. On one extreme highly fine-grained quality configurability may be achieved by having each group contain one bit. However, doing so presents significant control overheads to store and decode the quality field within the instruction. Therefore, we perform bit-significance driven approximation, wherein we group bits based on their significance and associate a quality level with each group. Figure 5.8 shows an example illustrating how different quality levels are associated to the various bit groups. In this example, the $qField$ is

[2]While we consider the error metric as expected error rate for illustration, other metrics such as expected error magnitude is also possible.

an 8-bit vector with each 2-bit group representing a different quality level (denoted by the different shades in color). Consider a data vector comprising of scalar elements S_0, S_1, ..., S_N with each scaler element having a bitwidth of 12, being read using the proposed load instruction. Accordingly, each quality level is associated with the corresponding 3 bits of S_0, S_1, ..., and S_N respectively, in a bit-significance manner as shown in the figure. In our experiments, we use 4 bit groups in the quality fields of load/store instructions.

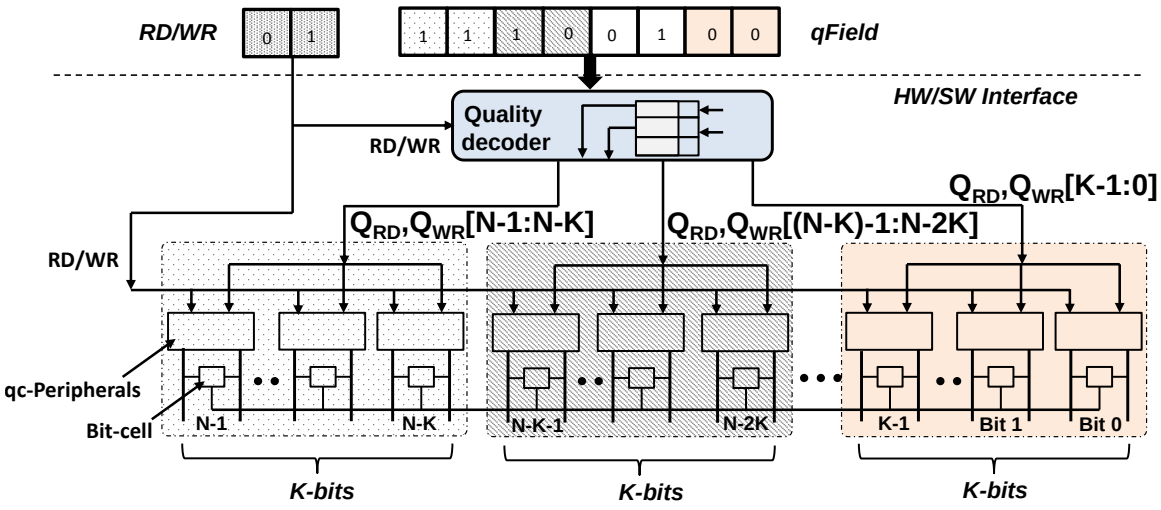


Fig. 5.9.: Conceptual overview of quality translation in a QcMem array

Quality translation in a QcMem array. Figure 5.9 provides a conceptual overview of quality translation in the QcMem array. The *quality decoder* takes the quality field ($qField$) and the read/write signal (RD/WR) as inputs and generates appropriate quality control signals ($Q_{RD}, Q_{WR}[N-1:\, N-K], ..., Q_{RD}, Q_{WR}[K-1:\, 0]$) for each group of K data bits. Depending on whether it is a read or write operation, the quality translation in the decoder happens differently. This is because the energy *vs.* quality trade-offs for both these operations vary significantly.

In summary, quality-aware load/store instructions are used to capture the quality requirements of memory operations at the instruction-level. Using appropriate

decoding logic, we translate these quality specifications into actual hardware knobs, thereby enabling us to exploit the capabilities of QcMem arrays for energy efficiency.

5.4.3 Auto-tuning instruction-level quality fields

Most applications, including the ones that are highly amenable to approximate computing, have a mix of sensitive data that is not tolerant to approximations, and resilient data that may be approximated. The ability to distinguish between resilient and sensitive data in memory is a major challenge in using approximate storage. Also, the quality field for each quality-aware load/store instruction in a program needs to be determined such that the energy benefits are maximized for a given output quality constraint. We propose an automatic tuning framework in order to address these challenges.

Algorithm 1 Tuning the quality knobs for approximation

Input: Application program and dataset: $Prog$, $DataSet$

Q-E trade-off curves for read/write: QE_c

Application quality target: Q_{target}

Output: Quality-fields for each memory instruction: $QFields$

1: **Begin**
2: $MemInst_{list}$: List of memory instructions in $Prog$
3: $QFields = 100\%$ $\forall$ instructions $\in MemInst_{list}$
4: **while** ($MemInst_{list} \neq \emptyset$) **do**
5: $E_{list} \leftarrow$ Energy of instructions in $MemInst_{list}$
6: I_{cand} = instruction with max(E_{List})
7: $I_{cand}.QField = QFields[I_{cand}]$ *// Get quality field*
8: $I_{cand}.FOM_{list}$: List of FOMs with Q-subfields relaxed
9: **for** each Q_{sub}: m bits in $I_{cand}.QField$ **do**
10: Q_{subNew} = *get_next_qlevel*(QE_c, Q_{sub})
11: E_{curr} = *get_energy*(QE_c, Q_{sub}, *LD/ST*)
12: E_{new} = *get_energy*(QE_c, Q_{subNew}, *LD/ST*)
13: $FOM = {}^{(E_{curr} - E_{new})}/_{(Q_{sub} - Q_{subNew})}$
14: $I_{cand}.FOM_{list} = I_{cand}.FOM_{list} \cup FOM$
15: **end for**
16: $I_{cand}.QField$ = update Q-subfield with max $I_{cand}.FOM_{list}$
17: $QFields_{new}$ = update $I_{cand}.QField$ in $QFields$
18: Q_{curr} = *get_app_quality*($Prog$, $DataSet$, $QFields_{new}$)
19: **if** ($Q_{curr} > Q\_target$) **then**
20: $QFields = QFields_{new}$
21: **else**
22: remove I_{cand} from $MemInst_{list}$
23: **end if**
24: **end while**
25: **return** $QFields$
26: **End**

Algorithm 1 uses a gradient descent approach to obtain the quality fields for all the memory instructions. The inputs are the application program and dataset ($Prog$, $DataSet$), the energy *vs.* quality trade-off curves for the different read and write schemes (QE_c), and a target quality constraint (Q_{target}) that bounds the degradation in application-level quality.

We first identify all the memory instructions in the application ($MemInst_{list}$) and initialize their quality fields to 100% (lines 2-3). Next, the algorithm iteratively finds out the appropriate quality fields for each instruction until none of the memory instructions in the application can be further approximated (lines 4-24). In each iteration, the following steps are performed. First, the energy consumed by each memory instruction in the application is estimated based on the energy model of the vector processor (line 5). We next identify the most energy-intensive memory instruction (I_{cand}) in the application (line 6) and introduce approximations in it. Since, a set of bits (m bits) in the quality field determines the significance of the error for a bit group (*i.e.* group of K data bits), we compute a figure of merit (FOM) for each of them using the following parameters (line 13): (i) the additional benefits in energy obtained by approximating the K bits (line 11 and 12), and (ii) the expected error rate (line 10). Depending on the instruction type (load/store), these parameters are directly obtained from the corresponding energy *vs.* quality trade-off curves. Next, each set of m bits (Q_{sub} fields) in the quality field of I_{cand} is ranked using its figure of merit (with the help of $I_{cand}.FOM_{list}$) and the highest one is chosen for target quality evaluation (line 16). If the quality constraint is met (line 19), the algorithm updates the quality fields for the candidate instruction (line 20) and proceeds to the next iteration (lines 4-24). On the other hand, if the target quality is violated, the instruction is removed from the $MemInst_{list}$ (line 22) and lines 4-24 are repeated.

Thus, the above approach enables us to automatically set the desired quality fields for memory instructions so as to maximize the overall energy benefits under a given output quality constraint.

5.5 STAxCache: Cache design with QcMem

In this section, we describe the spintronic approximate cache (STAxCache) architecture that allows for different levels of approximation to different parts of a program's memory address space. Figure 5.10 provides an overview of the STAxCache architecture. STAxCache consists of a data array realized using the proposed QcMem array, and a tag array that is not subject to approximation[3]. The data array is further composed of heterogeneous cache ways with varying retention levels (ways with lower retention time offer more energy-efficient writes). To allow control over which data is approximated and by how much, STAxCache also includes: (i) A *Quality table* that captures the data structure-level quality specifications of the application and (ii) A *Quality-aware cache controller* that regulates the quality of each cache access based on the entries in the quality table. The following subsections provide a detailed description of STAxCache.

5.5.1 Quality Table

STAxCache requires a mechanism to capture quality constraints in a manner that can be related to cache block-level reads (or writes). To this end, we introduce a quality table, as shown in Figure 5.10. Each entry in the quality table contains a memory address range and the desired quality for accesses to addresses within the range, *e.g.*, permissible magnitude of error that may be incurred when a location in the specified range is accessed. On each cache access, we compare the cache block address with the address ranges present in the table. If there is a match, we utilize the corresponding quality for reading (or writing) the block. We modulate the knobs present in the QcMem to achieve the desired read/write quality. In order to avoid significant impact on cache performance, we overlap the quality table look up and the subsequent quality decoding operation with the translation lookaside buffer (TLB)

[3]We focus on data array since it consumes a dominant part of cache energy. Furthermore, it is difficult to bound the impact of errors in the tag array.

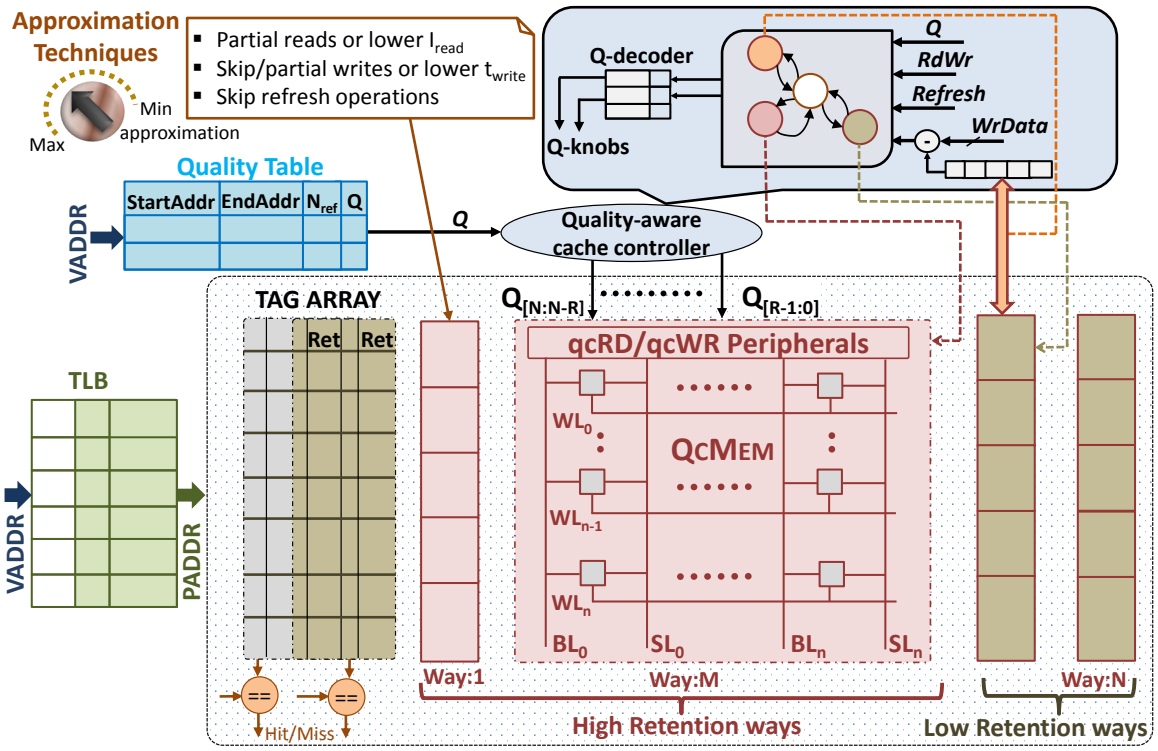


Fig. 5.10.: STAxCache organization

access and the following tag comparison operation. Our experiments indicate that the overheads of the additional quality table access on the cache performance is negligible ($< 0.3\%$).

5.5.2 Retention Approximations in STAxCache

Since most applications contain a mix of resilient and sensitive data, simply reducing the retention time for the entire cache is not acceptable. Therefore, we design a hybrid QcMem data array that comprises of both high retention and low retention ways, as shown in Figure 5.10. While we found only two levels of retention to be sufficient in practice (and desirable due to the lower design and manufacturing complexity), the concept extends to a larger degree of heterogeneity.

Refresh requirements. Cache blocks stored in the low retention ways are subject to a significant increase in the probability of errors beyond the retention time (T_{Ret}) owing to the exponential nature of retention failures [109]. Simply allowing retention errors is not always acceptable. While we preferentially allocate cache blocks with lower quality requirements to the low retention ways, data with very tight quality constraints (or data that cannot be approximated) may also be allocated to the low retention ways to ensure high cache utilization and low misses. Moreover, the lifetimes of cache blocks in low retention ways may vary considerably within and across applications. This imposes the need for periodic refresh operations, particularly when the lifetimes of the blocks are closer to (or exceed) T_{Ret}. Refreshing all the valid cache blocks in the low retention ways after each T_{ret} would ensure no retention errors, but leads to a significant number of refreshes. STAxCache addresses this issue by skipping refreshes for cache blocks that have been written due to a store instruction in the recent past. To enable this, we extend the tag array with only one retention bit per cache block to track the blocks stored in the low retention ways that have been written to or "self-refreshed" since the last refresh operation. Let us consider an example to demonstrate the proposed refresh mechanism. Figure 5.11(a) shows a 2-way set associative cache that consists of a high retention way and a low retention way. For this example, we need one retention bit per block as shown in the figure. Suppose at T = 0, two cache blocks ($B0$ and $B1$) are inserted in the low retention way. At T = $T_{Ret}/2$, the retention bits associated with all the cache blocks are checked. In case the bit is set to logic '0', we update it to logic '1', indicating that the block is due for a refresh operation in the next update cycle, *i.e.*, T = T_{Ret}, as shown in Figure 5.11(a) for $B0$ and $B1$. Next, suppose a write operation is performed on $B0$ between T = $T_{Ret}/2$ and T = T_{Ret}. In this case, the retention bit is reset to logic '0'. Hence, $B0$ no longer requires a refresh operation in the following update cycle. On the other hand, if the retention bit is set to '1', we perform the refresh operation for $B1$, as shown in Figure 5.11(a) at T = T_{Ret}, and reset the bit to '0'.

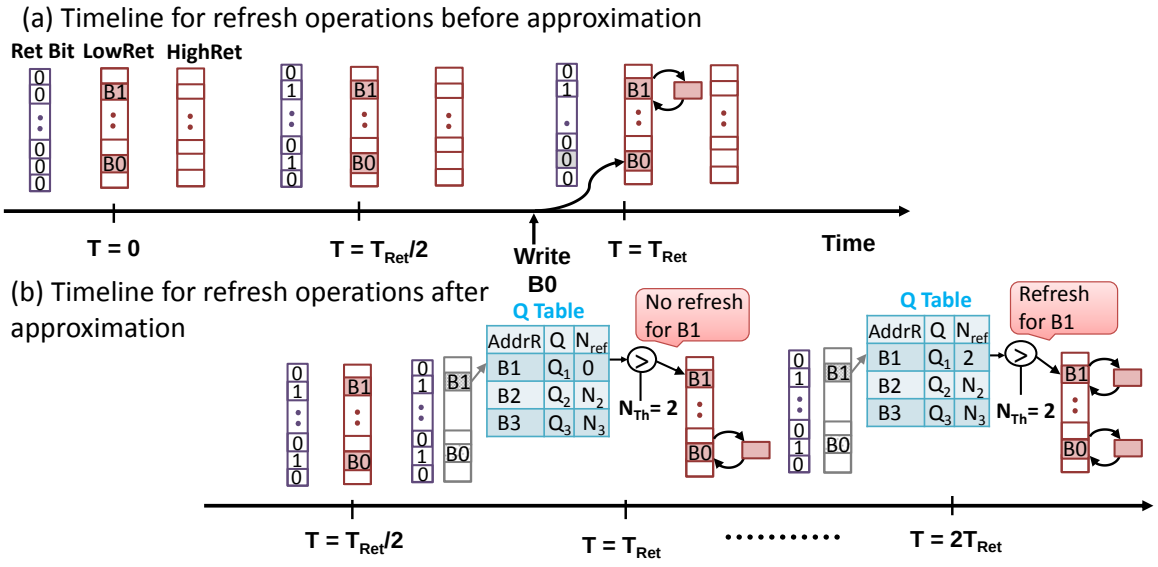


Fig. 5.11.: Timeline showing refresh operations

Approximations through skipped refreshes. Despite exploiting the self-refreshes to lower the refresh overheads, the energy consumed by the refresh operations still constitute a significant fraction of the total cache energy. In order to minimize the refresh energy further, we propose to skip refreshes to the blocks that are amenable to approximations. However, it is critical to have control over the retention errors introduced in the stored blocks as a result of the skipped refreshes. Towards this end, we extend the quality table with an additional counter (N_{Ref}) for each entry that tracks the number of refreshes skipped for a given address range on each update cycle. Figure 5.11(b) illustrates the proposed concept through a timeline for the cache organization discussed earlier. As shown in the figure, at T = T_{Ret} and T = $2T_{Ret}$, the addresses corresponding to $B0$ and $B1$ which are due for refresh (retention bits are set to '1'), are compared against the address ranges in the quality table. In case of a matching entry, the corresponding N_{Ref} is compared to a refresh threshold (N_{Th}) that is determined from the corresponding block-level quality constraint. If N_{Ref} exceeds (or equals) N_{Th}, we perform refreshes to each of the low retention blocks contained in the address range ($B1$ at T = $2T_{Ret}$), else we skip those refreshes ($B1$ at T = T_{Ret}

and T = $3T_{Ret}/2$) and increment N_{Ref} by 1. If a matching range does not exist in the table, the refresh operations are performed, as shown in the figure for $B0$ at T = T_{Ret} and T = $2T_{Ret}$.

5.5.3 Quality-aware cache controller

In this section, we describe the quality-aware cache controller (QACC) that utilizes the QcMem array to achieve read and write energy efficiency while preserving the quality constraints obtained from the quality table. Figure 5.10 provides an overview of the proposed QACC. The cache controller involves an additional control input (Q) that represents the desired quality for each cache access. QACC regulates quality in a bit-significance driven manner by dividing each word in the cache block into bit groups, and associating an approximation scheme for each group. This approach enables a fine grained control over the errors introduced during reads/writes. Specifically, a quality decoder takes the quality signal (Q) from the *quality table* and the read/write control signal ($RdWr$) as inputs and generates values of quality knobs ($Q[N-1:N-R], ..., Q[R-1:0]$) for each group of R bits of the words in the block. The QACC also takes the refresh control signal ($Refresh$) and produces the refresh threshold (N_{th}) based on the Q input, to determine when the refreshes can be skipped as discussed above. Since the energy *vs.* quality trade-offs widely vary across the different schemes, a systematic approach is required to obtain these knobs such that the energy savings are maximized for a given quality bound. In the following paragraphs, we describe how the quality knobs are obtained.

Read quality modulation. Figure 5.12 summarizes the approach used *at design time* to obtain the read quality knobs. Consider the surface plot of the block-level quality (Q_{blk}) for the different quality (and energy) configurations of the two schemes – partial read and reducing the read current (Q_1/E_1, Q_2/E_2), as shown in the figure. However, for a given block-level quality Q, there exist several (Q_1, Q_2) configurations that achieve the same quality (Q) although with significantly different energies, as

denoted by the Q plane. Therefore, we adopt a gradient descent approach to achieve the configuration that maximizes the energy benefits for a given Q. In each iteration, we rank the schemes based on the additional energy savings obtained by approximating a group of R bits (also referred as a bit group) for each word and the expected error introduced in the process. The expected error is estimated using a distribution of resilient data elements observed during reads. Next, we choose the scheme for the bit group with the highest ratio of energy benefits and expected error, and proceed to the next iteration until the block-level quality constraint is not violated. The above steps are repeated for each quality level supported by the cache.

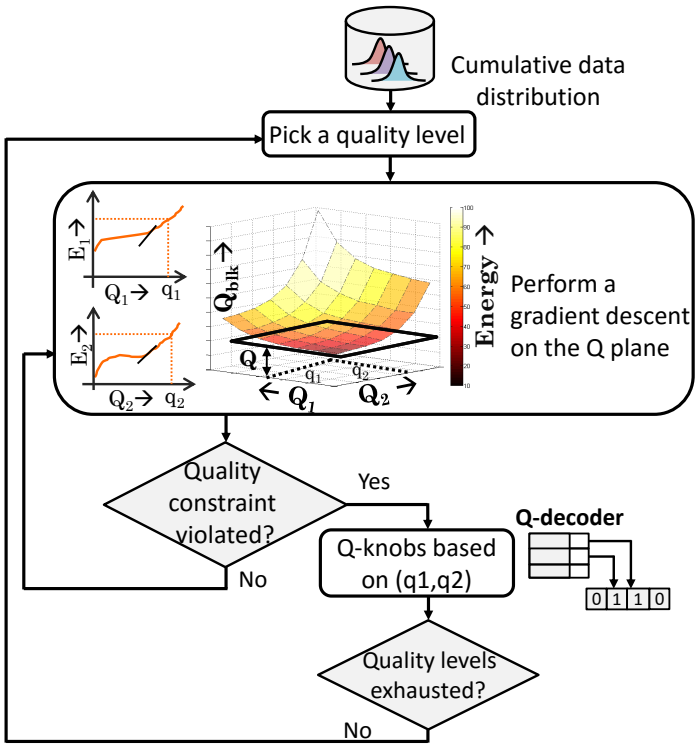


Fig. 5.12.: Read quality modulation

Write quality modulation. Figure 5.13 illustrates how the quality knob is obtained for writes in the proposed QACC. We utilize a *hybrid design time/runtime* approach to obtain the write quality knobs in the proposed QACC. We first determine if the block-level quality constraint is satisfied when a write to the block is skipped

altogether (*i.e.*, the stale value is retained). If the constraint is not violated, we skip the write operation; else, we utilize the pre-computed knobs obtained via a similar approach as the one discussed for reads.

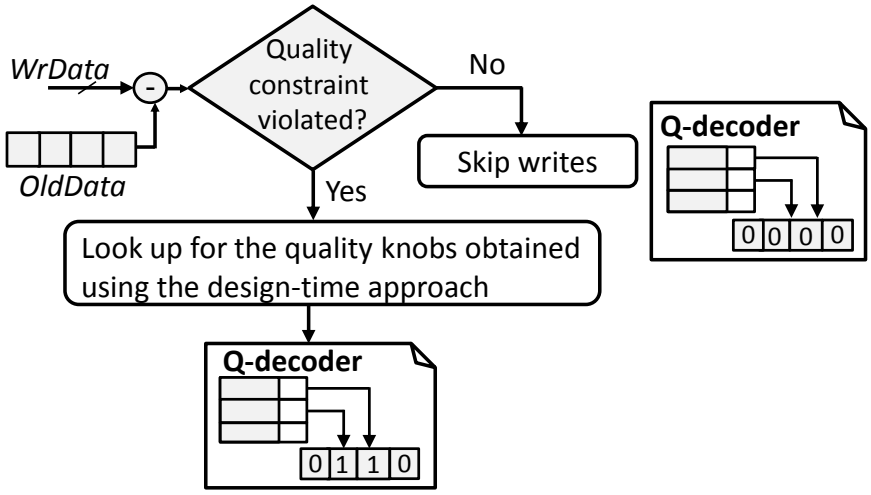


Fig. 5.13.: Write quality modulation

5.5.4 Cache insertion and replacement policy

We introduce suitable enhancements to the cache insertion and replacement policies to exploit the heterogeneous cache ways for energy efficiency. Specifically, we employ an insertion policy that is driven by the write intensity of the incoming block. We utilize a software-based approach similar to [110] to identify the write intensive blocks. The identified write intensive blocks are inserted in the low retention ways, whereas the other blocks can be inserted either in a low retention or high retention way depending on the block being evicted. We therefore introduce a modified least recently used (LRU) policy that evicts the LRU block within the low retention ways for a write intensive incoming block and uses the regular LRU policy for the remaining blocks.

5.5.5 ISA extension

In order to expose the intrinsic resilience of applications to the STAxCache architecture, we introduce a new instruction in the ISA that is used to identify resilient regions in the application's memory space, and also specify the desired quality bounds for them. Using this instruction, we update the proposed quality table, which is in turn used to regulate the accuracy of cache accesses.

Instruction format. Equation 5.4 shows the instruction format of the proposed instruction. As shown in the example, the instruction specifies the start and the end addresses associated with the resilient memory region (provided through registers R_{StAddr} and $R_{EndAddr}$) along with the maximum error bound that can be tolerated during read, write or refresh operations.

$$\begin{aligned} &\text{Format: } \textbf{Opcode} \text{ Reg1, Reg2, Reg3} \\ &\text{Example: } \textbf{ldQTable } R_{StAddr}, R_{EndAddr}, R_{Quality} \end{aligned} \tag{5.4}$$

5.5.6 Software support for STAxCache

To reduce the programmer effort required to utilize STAxCache, we propose a software interface that identifies the data structures that are resilient to approximation within a program, and an auto-tuning framework that appropriately modulates the data structure-level error constraints so as to achieve the target application quality. Note that the auto-tuning framework employed in Section 5.4.3 is not applicable for STAxCache since we evaluated STAxPad in the context of a vector processor memory hierarchy. In this design, the number of memory instructions amenable to approximations were much smaller compared to the memory instructions for a general purpose processor, thereby allowing an exhaustive exploration of the entire set of memory instructions.

Interface for approximate memory regions. We propose a programmer interface that can be used to specify resilient data within a program. Specifically, we introduce

new functions that capture the permissible error bound that can be tolerated in various cache operations for a given address range. Figure 5.14 shows an example code snippet that uses one such function to specify the data structure-level quality requirements in a k-means clustering application. In this example, an input array (*points*) is identified for approximations using the proposed function `set_axLevel`. The function `set_axLevel` takes the start and the end address corresponding to the *points* array and the tolerable error magnitude as inputs, and creates an entry in the quality table using the proposed instruction. This enables a simple interface for the programmer to specify quality targets for resilient data structures.

```
int main() {
   int* points = (int*) malloc (NUM_DATA * INT_SIZE);
   set_axLevel (points, (points + NUM_DATA), Q);
   read_points (points, NUM_DATA);
   int* means = (int*) malloc (NUM_MEANS * INT_SIZE);
    ……..
    ……..
    while (converges) {
        find_clusters (points, means, clusters);
        compute_means (points, means, clusters);
   }
}
 void set_axLevel(int startAddr,  int endAddr, int err_mag) {
     __asm__ ( "movl  %1, %%eax ;
                      movl  %2,  %%ebx ;
                      movl  %3,  %%ecx;
                      ldQTable %%eax, %%ebx, %%ecx"
                      :  /* no outputs */
                      :  "a" (startAddr) , "b" (endAddr), "c“ (err_mag)
}                  );
```

Fig. 5.14.: K-means clustering application for STAxCache

Auto-tuning Framework. Algorithm 2 provides a conceptual overview of the auto-tuning framework that automatically achieves the appropriate quality constraints for each data structure (DS) that is resilient to approximations. The proposed framework takes as inputs the application program (*Prog*), a representative training dataset

($TrData$), a target output quality (Q_{target}), and a list of data structures that can be approximated (DS_{list}).

Algorithm 2 Tuning the DS-level quality constraints

Input: Application program and training data: $Prog$, $TrData$

Application quality target: Q_{target}

Error-resilient data structures: DS_{list}

Output: List of DS-level constraints: QDS_{list}

1: **Begin**
2: δ: Granularity at which DS quality is modulated
3: $QDS_{list} = 100\%$ $\forall$ data structures $\in DS_{list}$
4: $E_{list} \leftarrow$ count accesses to each DS in DS_{list}
5: **while** ($DS_{list} \neq \emptyset$) **do**
6: DS_{cand} = DS with max(E_{List})
7: $QDS_{list}[DS_{cand}] = QDS_{list}[DS_{cand}]$ - δ
8: $Q_{app} = get\_app\_quality(Prog, TrData, QDS_{list})$
9: **if** ($Q_{app} < Q_{target}$) **then**
10: $QDS_{list}[DS_{cand}] = QDS_{list}[DS_{cand}] + \delta$
11: remove DS_{cand} from DS_{list}
12: **end if**
13: **end while**
14: **return** QDS_{list}
15: **End**

The algorithm iteratively obtains the DS-level quality constraints until none of the resilient data structures can be approximated any further (lines 5-13), while simultaneously ensuring the output quality bound is satisfied. E_{list} records an estimate of energy consumed by each DS by counting the corresponding number of accesses in the program (line 4). In each iteration, the following steps are performed. First, the most energy-intensive DS (DS_{cand}) is identified in the program (line 6). Next, the

quality of DS_{cand} is reduced by δ, the smallest granularity chosen to modulate the DS-level quality (line 7). The resulting application with the modulated constraint is then evaluated subject to the target quality constraint (line 8). If Q_{target} is violated (line 9), the algorithm reverts the current quality modulation (line 10) and removes DS_{cand} from the DS_{list} (line 11).

Thus, the above approach minimizes the programmer effort by automatically setting the appropriate data structure quality that yields energy benefits for each application with a target output quality constraint.

5.6 Experimental Methodology

In order to evaluate the proposed concepts, we developed a device-circuit-architecture modeling framework. In this section, we describe the modeling framework and the application benchmarks used in our evaluations.

MTJ Device/Array-level modeling. The STT-MRAM bit-cell was characterized in SPICE using a 32nm CMOS transistor model and a SPICE-compatible PMA MTJ device model based on self-consistent solution of Landau-Lifshitz-Gilbert (LLG) magnetization dynamics and Non-Equilibrium-Green's Function (NEGF) electron transport [111] with the device parameters shown in Table 5.1 [112]. To estimate the array-level latency and energy of the QcMem array, we use the bit-cell characteristics as technology parameters in a modified version of CACTI [113] that is designed for STT-MRAMs. In our analysis, we also consider the additional circuit-level overheads of the quality-configurable peripherals.

STAxPad evaluation. In order to evaluate the benefits of the STAxPad at the system-level, we utilized a cycle accurate simulator for the vector processor discussed in section 5.4.1. We modified the simulator to model a 1MB on-chip QcMem array and evaluated both the application energy benefits and the output quality. In order to estimate the overall application energy (*i.e.*, total processor and memory energy expended while executing the application), we synthesized the RTL implementation

Table 5.1.: MTJ device parameters

Saturation Magnetization (M_s)	850 emu/cm^3
Damping Factor (α)	0.028
Nominal MTJ Dimension	64nm x 64nm x 1nm
Oxide Thickness (t_{ox})	1nm
Energy Barrier	64 K_BT/16 K_BT
Temperature	300K
Assumed Variation (σ/μ)	t_{ox} = 2%, MTJ area = 5%, transistor V_T = 5%

of the processor using Synopsys Design Compiler to a 45nm IBM technology library and used Synopsys Power Compiler to estimate the power consumption of the mapped netlist. We also obtained memory access traces from the simulator and used them along with the read/write energies obtained from the modified CACTI tool to compute the total memory energy.

Table 5.2.: STAxCache system configuration

Processor Core	x86, out-of-order processor, 2 GHz
L1 I/D-cache	32KB/64KB, 2 way-set associative, 64B line size
L2 unified cache	8MB shared, 8 way-set associative, 64B line size
Cache latency	L1: 2-cycle, L2 read: 10-cycle, L2 write: 15-cycle

STAxCache evaluation. We model the STAxCache architecture in the architectural simulator gem5 [97] and use it to evaluate the energy benefits and the impact on application-level quality. Table 5.2 shows the processor configuration used in our evaluation. In our experiments, we consider 4 low retention ways (with an energy barrier of 16 K_BT) for the L2 cache. We use the cache access traces from the simulator along with the array-level energies obtained from the modified CACTI tool to estimate the L2 cache energy. We also account for the overheads in energy and latency associated with the proposed quality table hardware in our experiments.

Benchmark applications. Table 5.3 lists the applications, the underlying algorithm and the datasets that were used to evaluate our proposal. The quality metric used to evaluate the output quality in each benchmark is also provided.

Table 5.3.: Application benchmarks for approximate storage

Application	Algorithm	Dataset	Quality metric
Digit Recognition (SVM)	Support vector machines	MNIST	Classification accuracy (fraction of inputs correctly classified)
Text Classification (TEXT)	Support vector machines	REUTERS	
Digit Classification (CNN)	Convolutional neural networks	MNIST	
Character Recognition (OCR)	K-nearest neighbors	OCR digits	
Web Page Classification (Web)	K-nearest neighbors	Web	
Eye Detection (GLVQ)	Generalized learning vector quantization	YUV faces	
Protein Structure Classification (MLP-P)	Multi-layer perceptron	Protein	
Object Recognition (MLP-C)	Multi-layer perceptron	CIFAR-10	
Image Segmentation (IMGSEG)	K-means clustering	Berkley dataset	Mean cluster radius
Optical Character Clustering (DIGITS)	K-means clustering	OCR digits	

5.7 Experimental Results

This section presents the results of various experiments that quantify the benefits of QcMem.

5.7.1 Energy benefits of STAxPad

Figure 5.15 illustrates the improvement in total application energy achieved using STAxPad in a vector processor memory hierarchy for different target output quality (classification accuracy) constraints. The application energy is normalized to the completely accurate case which has the normal memory array, in which no additional peripheral logic is introduced for quality-configurability. Across all bench-

marks, QcMem achieves energy benefits ranging from 9%-30% for virtually no loss ($< 0.5\%$) in application quality. When the quality constraints are relaxed to $< 2.5\%$ and $< 7.5\%$, the energy benefits further increase to 10%-32% and 14%-34% respectively. On average, STAxPad obtains 19.5%, 22.3% and 28% improvement in the total application energy for the different quality constraints.

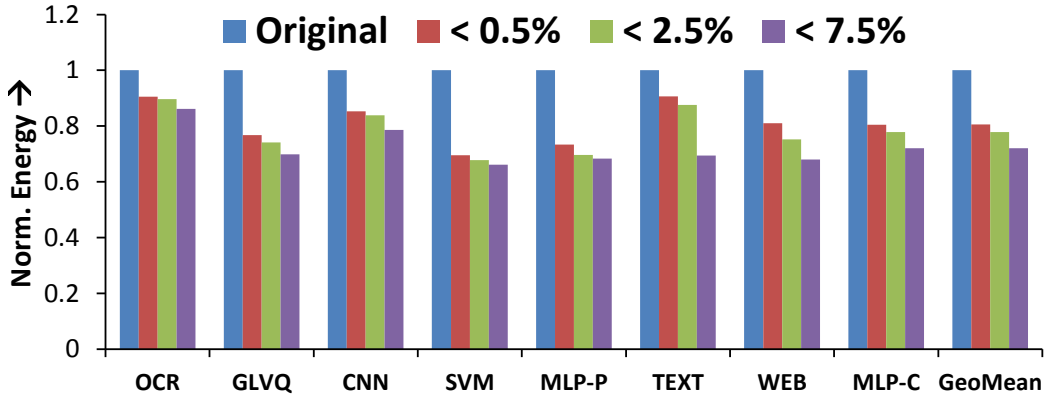


Fig. 5.15.: Improvement in system energy with STAxPad

We next discuss the different application energy components that contribute to the overall energy. Across all benchmarks, we found that the memory consumes 49.7% (on average) of the total application energy, with the read and write energies being 28% and 22%, respectively. Although write operations consume significantly higher energy compared to reads in STT-MRAMs, the higher numbers of reads in our applications results in the overall energies being comparable. Figure 5.16 shows the memory energy and its read and write energy components for all benchmarks at various application quality constraints. The energy is normalized to the memory energy consumption in QcMem with no approximations introduced. Across all benchmarks, we achieve 40% improvement in memory energy on average for a negligible loss in classification accuracy. Correspondingly, we obtain 1.48× and 1.76× improvement in read and write energy, thereby demonstrating the effectiveness of the proposed techniques in approximating both read and write operations. For relaxed constraints ($< 2.5\%$ and $< 7.5\%$), we obtain even more substantial improvements in the overall memory energy

(45% and 55% respectively). This corresponds to 1.62× and 1.7× improvement in read energy and 1.88× and 2.37× improvement in write energy.

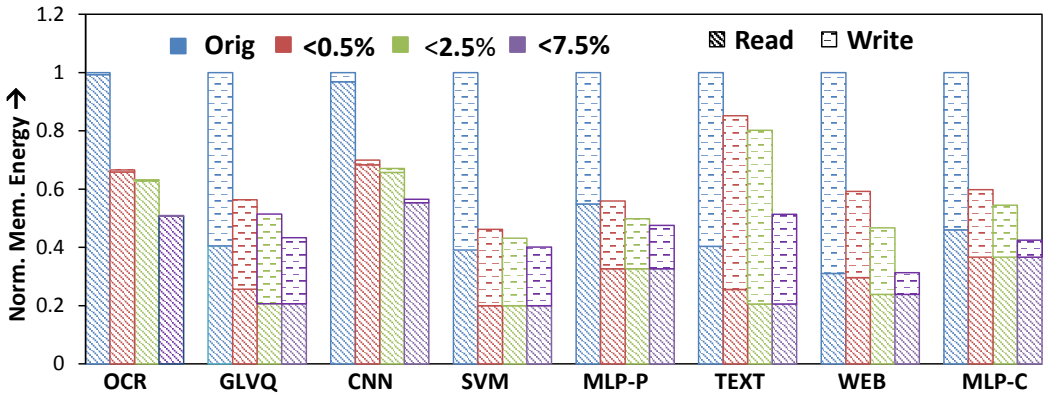


Fig. 5.16.: STAxPad energy breakdown

5.7.2 Energy benefits of STAxCache

Figure 5.17 shows the normalized L2 cache energy obtained using the proposed architecture at different application-level output quality targets. The L2 cache energy is normalized to an accurate cache with heterogeneous ways[4]. The energy benefits range from 1.15× to 1.84× for a negligible loss (< 0.5%) in application-level quality over all benchmarks. For a more relaxed quality requirement (< 3.5% degradation in application output quality), the benefits further extend to 1.5×-2.32×. On average, STAxCache achieves 1.44× and 1.93× improvement in energy at the two quality levels, respectively.

We next present a breakdown of the different energy components, *viz.* read, write, refresh and leakage, that contribute to the overall cache energy. Figure 5.18 shows the breakdown for all benchmarks at the two output quality constraints (< 0.5% and 3.5% loss in quality). For the baseline, across all benchmarks, we observe that the read, write, and refresh energies respectively constitute 26%, 61% and 10% of

[4]We focus on an STT-MRAM cache designed with low and high retention ways since it consumes lower energy than a standard STT-MRAM cache.

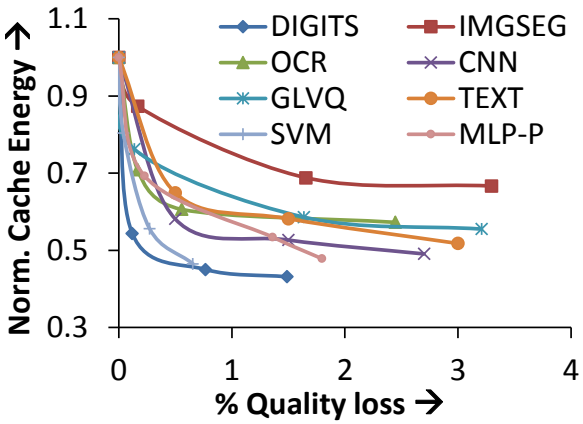


Fig. 5.17.: Improvement in energy using STAxCache

total cache energy, with refresh overheads being as high as 48% of the total energy in some cases. On average, we achieve 1.03× and 1.56×, improvement in read and write energy, for $< 0.5\%$ loss in output quality. Note that, for a subset of benchmarks (DIGITS, IMGSEG, OCR, and GLVQ), the read energy increases beyond the baseline. This is due to the write skipping approximation technique, that involves an additional read operation. In these benchmarks, such read operations dominate over the original reads, thereby increasing the total read energy, but enabling an even higher savings in write energy. For a relaxed quality bound ($< 3.5\%$), we obtain greater benefits in read and write energy, 1.29× and 2.22×, respectively. For benchmarks with significant refresh operations (CNN, IMGSEG), we achieve 1.68× and 2.02× improvement in refresh energy for the tight and relaxed bounds respectively, illustrating the effectiveness of the proposed design in mitigating the refresh overheads through approximations.

5.7.3 Impact on system performance with STAxCache

Figure 5.19 compares the IPC (instructions per cycle) for STAxCache at both output quality constraints to the baseline design having no approximations. On average, STAxCache degrades cache performance by 1.9% and 1.7% over the baseline

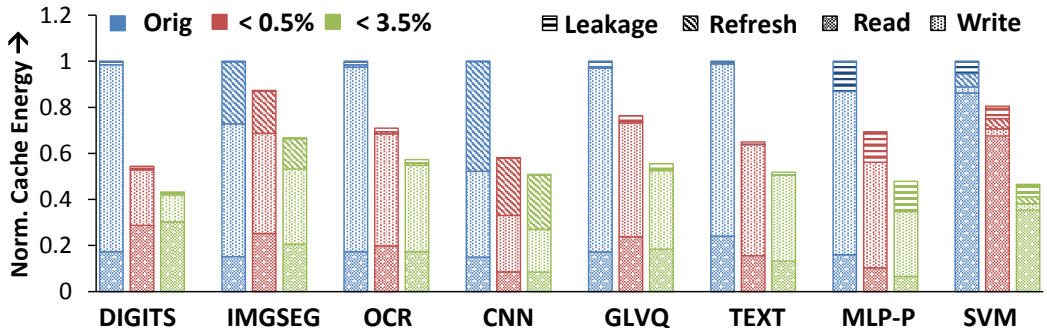


Fig. 5.18.: Energy breakdown for STAxCache

design for the two quality levels. This degradation in performance is mainly due to two factors: (i) the additional latency for writes that are not skipped (since we perform a read to check whether the write can be skipped), and (ii) the overheads of accessing the quality table during each cache access and update cycle.

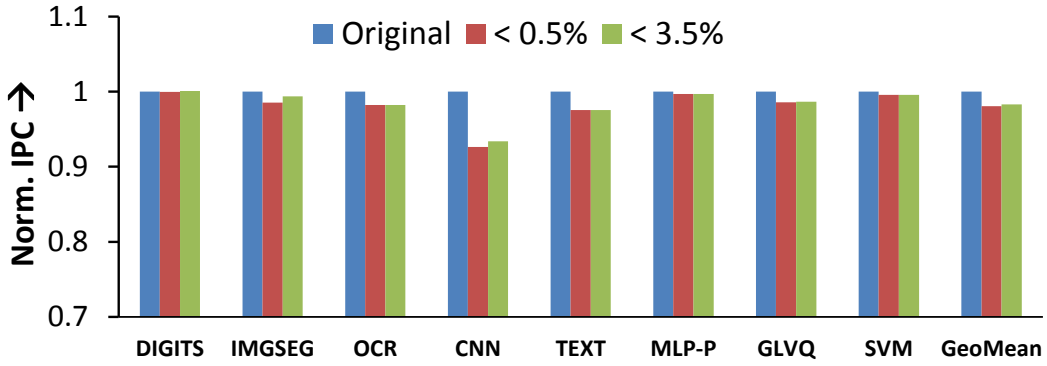


Fig. 5.19.: Performance trend with STAxCache

5.7.4 Comparison with uniform approximation

We evaluate the effectiveness of providing fine-grained quality control for different groups of bits within each word by comparing it against a simpler uniform approach where all the data bits in a word are read/written with the same quality. Figure 5.20 shows the error *vs.* quality trade-offs obtained for three different applications when executed on the vector processor-based platform. For all the applications, the energy

benefits obtained using the proposed significance-based quality specification framework are substantially better than the uniform approach, underscoring the need for the proposed quality controls.

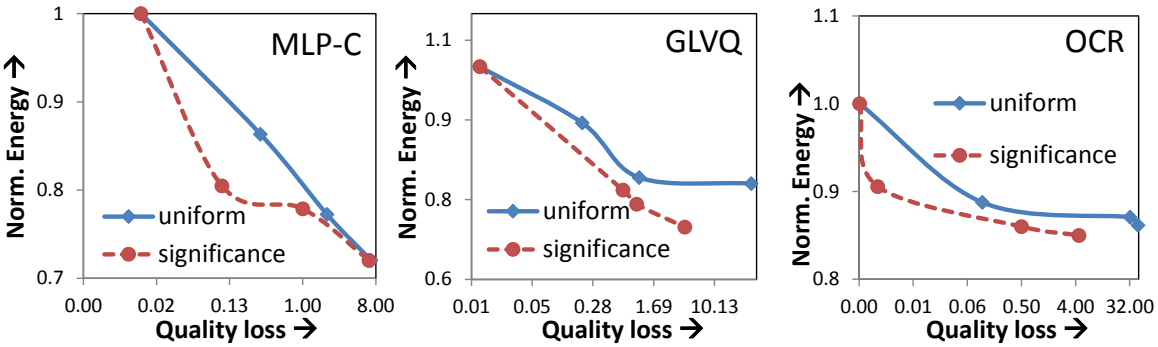


Fig. 5.20.: Benefit of significance-based approach

5.7.5 Comparison with a single approximation scheme

We next evaluate the effectiveness of utilizing a combination of approximation techniques as in STAxCache. We compare it against the scheme that only uses partial reads, along with skipped and partial writes. Figure 5.21 shows the energy and output quality trade-off obtained with different data-structure level quality constraints for two applications. We obtain a superior energy *vs.* output quality trade-off with a combination of techniques, justifying the proposed approach.

5.7.6 Energy vs. quality trade-off

We now explore the application-level energy *vs.* quality trade-off that can be achieved by varying the error bounds associated with the quality-aware load/store instructions proposed in the context of the vector processor. Figure 5.22 shows the $Q - E$ trade-off for two applications – SVM and MLP-P. In both cases, we achieve substantial improvements in energy at negligible loss in quality for smaller instruction-level error bounds. However, beyond a point, the energy benefits taper down and

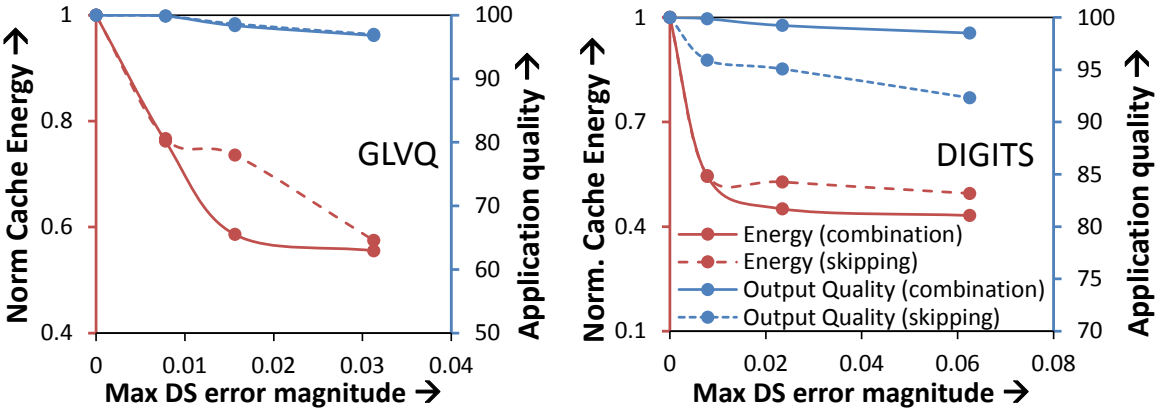


Fig. 5.21.: Comparison of energy benefits over skipping scheme

further introducing errors in memory instructions results in a sharp degradation in output quality.

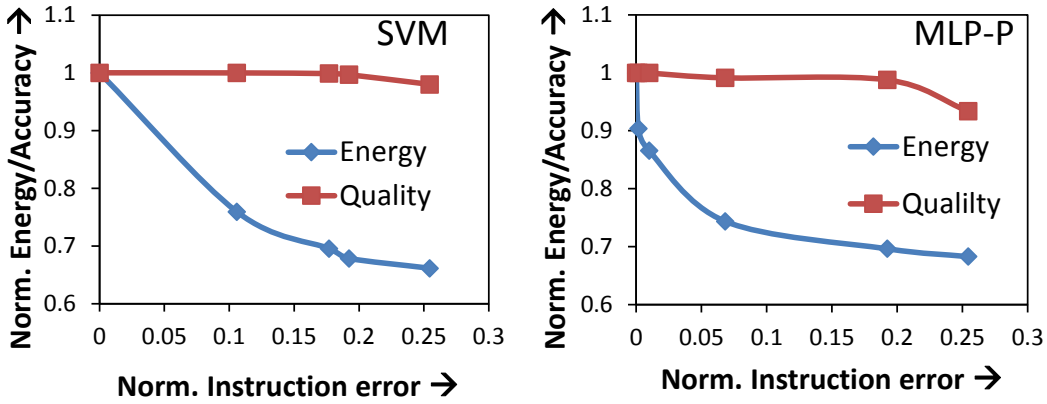


Fig. 5.22.: Energy *vs.* quality trade-off for two applications by varying instruction-level error bounds

5.8 Summary

The energy-inefficiency of STT-MRAMs pose a major challenge to their adoption as a potential post-CMOS memory technology. In this chapter, we utilized approximate storage that leverages the intrinsic resilience of applications to improve the read

and write energy efficiency of STT-MRAMs. We identified various mechanisms at the circuit and architecture level that offer disproportionate energy benefits at the cost of small read, write or retention errors. We used these mechanisms to design a quality-configurable memory array that can store data at varying accuracy levels based on the application requirements. We evaluate the proposed memory array as a scratchpad for a vector processor and as an L2 cache for a general purpose processor. Our experiments on a wide range of recognition and search applications show that we can achieve substantial energy benefits for negligible loss in quality.

6. RECONFIGURABLE CACHE ARCHITECTURE USING DWM TAPES

6.1 Introduction

In the past several decades, the integrated circuit industry has witnessed a continuous surge in the demand for on-chip memory due to increasing data-set sizes and the widening processor-memory gap. A growing portion of chip area and energy consumption is expended in caches. Consequently, emerging memory technologies such as spintronic memories that promise high density and low leakage power are of great interest in cache design.

As described in Chapter 1, DWM is a spintronic memory technology that achieves very high density and improved energy efficiency compared to CMOS and other emerging memory technologies [6, 7]. This has kindled great interest in using DWMs to realize caches both in the context of general purpose processors [8, 23, 24, 26, 57] and domain specific accelerators such as GPUs [21, 25, 58]. DWMs have a unique tape-like structure that achieves very high density by packing several (∼10s-100s of) bits into the domains of a ferromagnetic nanowire [6]. However, this structure also leads to serialized accesses to the bits in each bit-cell via shift operations, resulting in higher access latency. Fig. 6.1 shows the trade-off between cache area and average access latency with increasing bits per tape for PARSEC and SPLASH benchmarks. As the number of bits per tape increases, we observe a significant reduction in cache area. However, this also increases the number of shift operations, resulting in increased average access latency. This increased access latency for larger number of bits per tape is a major challenge in the design of DWM-based caches.

In this thesis, we propose DyReCTape, a reconfigurable DWM-based cache that maximally exploits the density benefits of DWM while reducing the performance im-

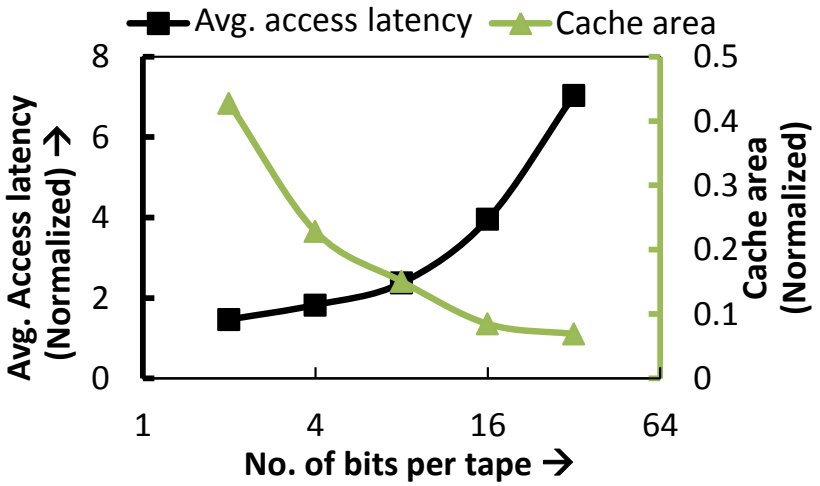


Fig. 6.1.: Average latency vs. area trade-off

pact arising from shift operations. The DWM structure provides a natural mechanism to reconfigure cache size by varying the active number of bits per tape. DyReCTape leverages this unique capability of DWMs to dynamically modulate cache capacity and latency in order to improve overall system performance. In doing so, it considers the varying cache access patterns and sensitivity of applications to cache size and latency.

The design of DyReCTape poses two challenges: (i) the reconfiguration mechanism should track the performance sensitivity of applications to varying cache size as well as shift latency, and should respond to the dynamically changing memory access patterns within and across applications, and (ii) transitioning across different configurations of varying cache sizes involves data migration that leads to considerable energy and performance overheads. We address these challenges through suitable optimizations in DyReCTape.

In summary, the key contributions of this work are as follows:

- We propose the concept of a dynamically reconfigurable DWM-based cache that mitigates the performance penalty from shifts by modulating the bits per tape based on workload characteristics.

- We present a reconfigurable cache architecture consisting of (i) an address remapping scheme that seamlessly supports different logical-to-physical mappings of cache blocks for various cache configurations, and (ii) a history-based reconfiguration policy that dynamically tracks cache statistics such as the incurred penalties due to shifts and misses, and based thereupon, adapts cache capacity to the varying requirements within/across applications.

- We propose two key optimizations to further improve the performance of DyReCTape: (i) In order to minimize the overheads associated with data migration during reconfiguration, we propose a lazy migration policy by leveraging the non-volatility of DWMs. (ii) In order to further boost the performance, we utilize the inactive portion of cache after reconfiguration as a victim cache.

- We perform a detailed evaluation of DyReCTape using a device-to-architecture modeling framework. Across benchmarks from the SPLASH and PARSEC benchmark suites, DyReCTape achieves a 19.8% and 11.7% improvement in performance on average over a traditional SRAM cache and a static DWM-based cache, respectively.

The rest of the chapter is organized as follows. Section 6.2 describes the DyReCTape architecture along with various optimizations explored to further improve performance. Section 6.3 explains the experimental methodology used to evaluate DyReCTape, and the results are presented in Section 6.4. Finally, Section 6.5 summarizes the chapter.

6.2 DyReCTape Architecture

Fig. 6.2 shows the overall architecture of DyReCTape. We follow the basic DWM-based cache organization described in [8, 24] — the tag array is designed with 1-bit DWM cells to avoid variable latency tag lookups, and the multi-bit DWM-based data array is organized into randomly addressable tape clusters [8]. We assume a bit-

interleaved mapping of cache blocks to each tape cluster, such that a cache block can be accessed in parallel after the appropriate number of shift operations is applied to all the tapes in a tape cluster. We refer the interested reader to [8, 24] for further details, and focus on the architectural enhancements for reconfiguration.

For reconfiguration, DyReCTape also includes: (i) *address remapping logic* that supports different logical-to-physical mappings for various cache configurations, (ii) *data migration logic* that controls the movement of cache blocks while transitioning across multiple cache configurations, and (iii) a *reconfiguration controller*, which uses cache statistics such as number of shifts incurred and miss rate to initiate cache reconfiguration. In the following subsections, we provide a detailed description of the DyReCTape architecture and different optimizations explored to maximize performance.

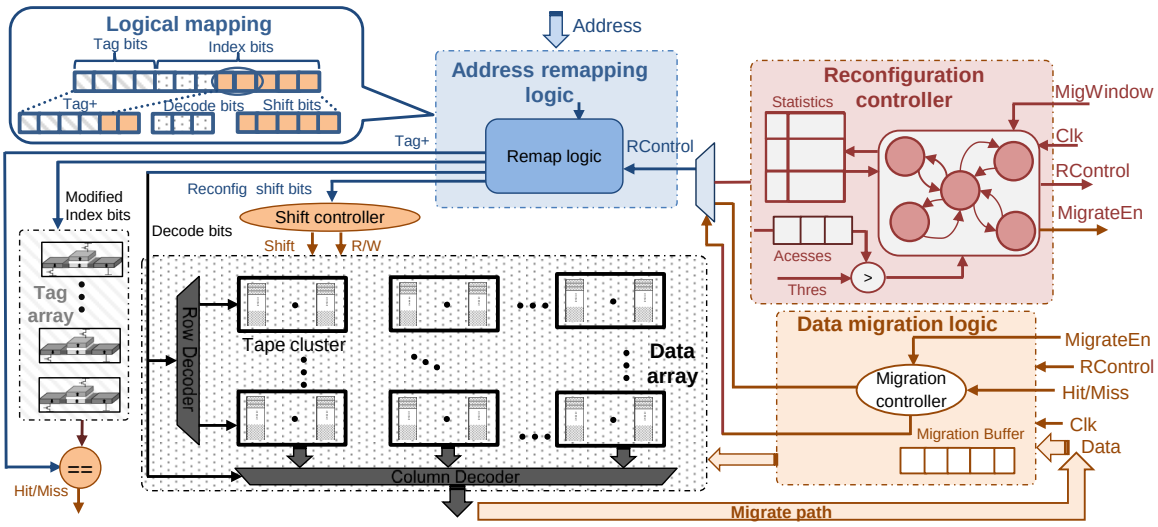


Fig. 6.2.: DyReCTape organization

6.2.1 Reconfiguration mechanism

DyReCTape involves dynamically varying the cache capacity by modulating the bits per tape. Let us consider an example to demonstrate how the reconfiguration is

achieved. Fig. 6.3(a) shows a 32 entry direct-mapped cache that consists of 4 tape clusters with each cluster storing 8 cache blocks (8 bits/tape). In order to access a cache block in this design, the index bits are decomposed into decode and shift bits. The decode bits are used to select the tape cluster to which a cache block is mapped, and the shift bits are used to identify the location of the cache block within the tape cluster. For this example, we need 2 decode bits for identifying the cluster number and 3 shift bits for accessing the cache block within a cluster, which is also shown in Fig. 6.3(a). Suppose we wish to reduce the cache capacity by half, *i.e.*, reconfigure the cache as a 16 entry direct-mapped cache, we reduce the number of cache blocks stored per tape cluster to 4 (4 bits/tape). Fig. 6.3(b) shows the reconfigured cache state with the inactive portion of the cache highlighted in gray. We choose such a reconfiguration strategy because: (i) it lowers the number of shifts thereby improving cache latency, and (ii) it retains the spatial locality associated with data blocks residing in the active region of cache. We now require 2 decode bits and 2 shift bits for accessing a cache block in the reconfigured state. Furthermore, determining hits/misses would require a wider tag (Tag+) which is composed of the original tag bits and the MSB of the original shift bits. In summary, it is necessary to have a suitable address remapping scheme in order to access a cache block from different reconfigured states.

Address remapping logic: In this work, we devise an addressing mechanism that can seamlessly support the different logical-to-physical mappings of cache blocks that manifest under various cache capacity states in a reconfigurable cache. This logical mapping is shown in Fig. 6.2. The address remapping logic automatically performs such a translation of an input address depending on the current cache configuration. As illustrated in Fig. 6.2, the inputs to the address remapping logic are the address for the cache block to be accessed and a control signal indicating the current cache configuration. It produces the modified index bits used to access the tag array, the decode bits for choosing the tape cluster and the modified shift bits, used to access the cache block in the reconfigured tape cluster. In this addressing scheme, we design the tag array to support the widest tag possible, as determined by the smallest cache

capacity that is supported by DyReCTape, and use it to identify hits/misses for all possible cache sizes. Note that, the area and energy overheads to enable a wider tag array for our implementation are negligible and the redundant comparisons do not impact the correctness of tag lookups.

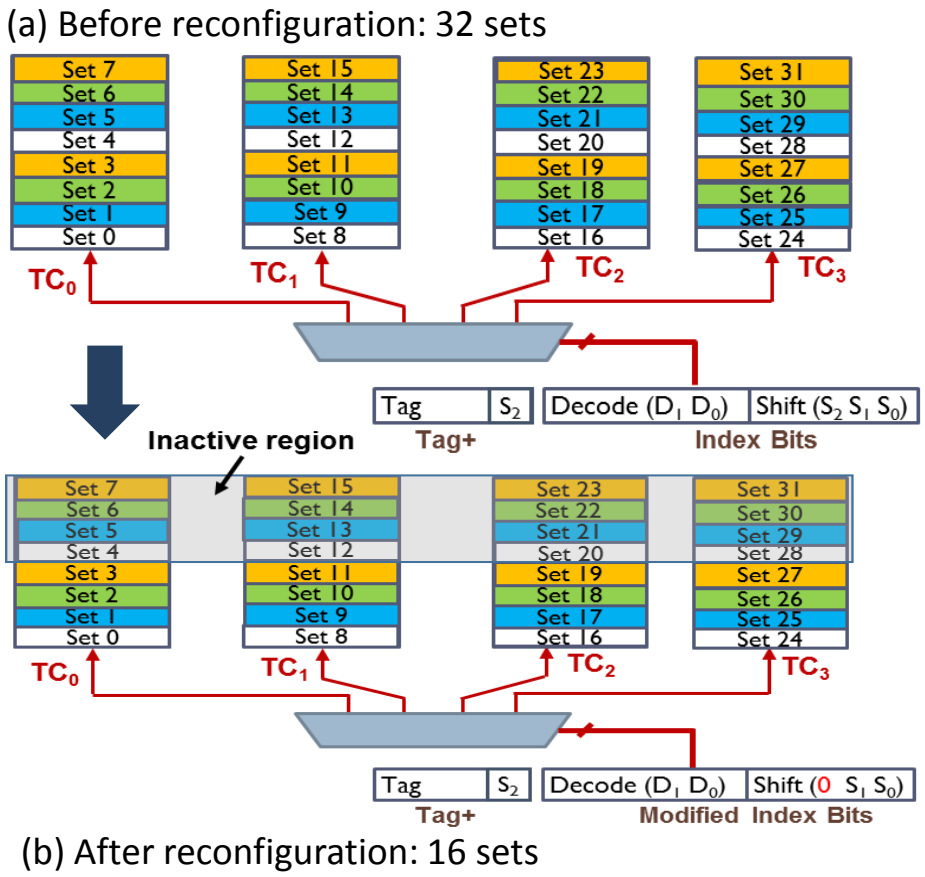


Fig. 6.3.: Reconfiguration overview

6.2.2 Reconfiguration policy

In this section, we describe the policy used to dynamically reconfigure the cache capacity in DyReCTape. Our reconfiguration policy considers the following cache statistics: (i) effective number of shifts, (ii) shift latency, (iii) miss latency, and (iv) effective miss rate. We divide the program execution into intervals that contain equal numbers of memory accesses, and compute the cumulative shift and miss penalties

in each interval in order to determine reconfiguration actions, *i.e.*, expand or shrink. Equation 6.1 and 6.2 shows the shift and miss penalties in the i^{th} interval of program execution.

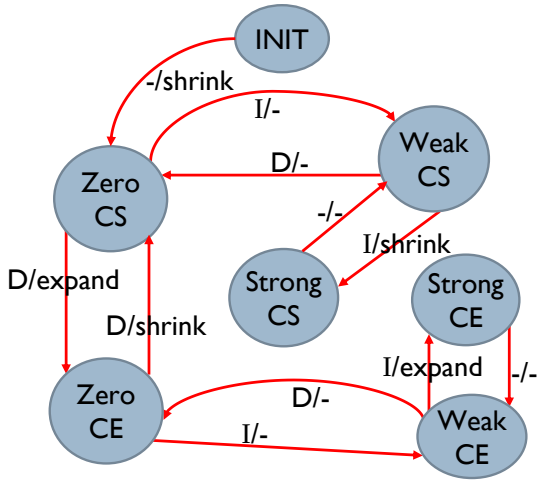
$$CSP_i = \sum_j H_{ij} * N_{S_{ij}} * T_{shift} \tag{6.1}$$

$$CMP_i = \sum_j (1 - H_{ij}) * T_{main} \tag{6.2}$$

In these equations, CSP_i denotes the cumulative shift penalty due to shift operations in the i^{th} interval. H_{ij} is a Boolean variable that indicates if the j^{th} access in interval i is a hit/miss, $N_{S_{ij}}$ is the numbers of shifts incurred during the j^{th} access and T_{shift} is the time required to shift the tape by one position. CMP_i represents the cumulative miss penalty in the i^{th} interval and T_{main} is the main memory latency. We compare the shift and miss penalties with a moving window average of their history to determine whether they are increasing or decreasing.

Fig. 6.4 illustrates the reconfiguration policy, which is based on the increasing/decreasing trend of CSP and CMP. Suppose that the prior reconfiguration action was an expand. If the increase in shift penalty outweighs the benefits of lower miss penalty in the current interval, we infer it as a degradation in cache performance and revert the expand action. On the other hand, if the decrease in cumulative miss penalty outweighs the corresponding shift penalty, the expand action is repeated. The conditions for improvement and degradation (shown in the table of Fig. 6.4) are exactly the opposite for a prior shrink operation. Note that, due to time-varying memory access characteristics, we might occasionally observe an effective increase or decrease in both shift and miss penalties. We therefore introduce thresholds (α, β) to determine the dominant factor amongst the two and use them to guide reconfiguration actions. In scenarios where we cannot determine the dominating penalty, the prior configuration is retained. In order to avoid frequent reconfiguration actions, we introduce three confidence states associated with every increase/decrease action, *viz.* zero, weak and strong. Depending on the history of improvements/degradation over their previous

average values, we transition into one of the three states, *i.e.*, improvement leads to an increase in confidence and degradation lowers the confidence in our current reconfiguration action. Note that, reconfiguration actions are only performed in the zero confidence or strong confidence states. Zero confidence states (Zero CE and CS) cause an increase/decrease of cache size without any prior confidence while strong confidence states (Strong CE and CS) result in reconfiguration actions based on prior history of improvements. The actions are reversed only in the zero confidence states when we observe a degradation in cache performance.



Expand (Shrink)	$(CSP_i - Avg.\ CSP) \geq \alpha*(CMP_i - Avg.\ CMP)$	D (I)
	$(CSP_i - Avg.\ CSP) \leq \beta*(CMP_i - Avg.\ CMP)$	I (D)

Zero confidence in shrink	Zero CS
Zero confidence in expand	Zero CE
Weak confidence in shrink	Weak CS
Weak confidence in expand	Weak CE
Strong confidence in shrink	Strong CS
Weak confidence in expand	Strong CE
I ➔ Improvement	D ➔ Degradation

Fig. 6.4.: Reconfiguration policy

6.2.3 Data migration logic

Reconfiguration involves an increase or decrease in cache capacity, which imposes the need for migration of cache blocks to ensure that they are stored at valid locations. We next discuss the different steps involved in this process and the proposed optimizations to reduce the associated overheads.

In DyReCTape, reconfiguration results in two kinds of actions: (i) shrink or decrease in bits per tape, and (ii) expand or increase in bits per tape. Shrink reduces the cache capacity, thereby rendering a portion of the cache inactive. Suitable actions

are needed to handle the blocks that are present in the inactive portion. A naive solution for this would require flushing all the dirty cache blocks in the inactive portion to memory and subsequently invalidating them. On the other hand, those blocks present in the active portion require no action since their logical mapping remains valid. This naive scheme presents the following overheads: (i) heavy data traffic to main memory while flushing the cache blocks, and (ii) increase in miss rate due to the invalidated data.

Next when we consider the expand operation, it involves an increase in cache capacity, thereby transitioning a portion of the cache from inactive to active state. Therefore, a block that was previously stored at one location in the cache can get mapped to a different location in the newly activated region of the cache. Addressing this issue would require stalling all incoming cache accesses and pro-actively migrating all the cache blocks to their respective valid locations in the expanded cache. This unavailability of the cache for a large number of execution cycles leads to a significant performance overhead.

Our experiments confirm that these overheads are indeed significant enough to overshadow the performance improvements obtained as a result of reconfiguration.

In order to mitigate the performance overheads associated with shrink and expand operations, we propose the concept of *lazy data migration* wherein data blocks are migrated over a migration time window to the appropriate location after reconfiguration. We next explain this scheme in detail for each of the two reconfiguration steps, *i.e.*, expand and shrink.

Lazy shrink: Fig. 6.5 demonstrates the lazy shrink policy with a timeline showing the decrease in bits/tape (*i.e.* capacity) over the migration window (denoted *MigrationWindow*). Before reconfiguration (at T = $T_{initial}$), the cache utilizes N bits per tape. At T = $T_{Reconfig}$, the reconfiguration action is initiated with a reduction in bits per tape. This renders a part of the cache inactive for normal cache operations. We note that the data already present in these locations can be retained without any overhead due to the non-volatile nature of DWMs. Based on this observation, the

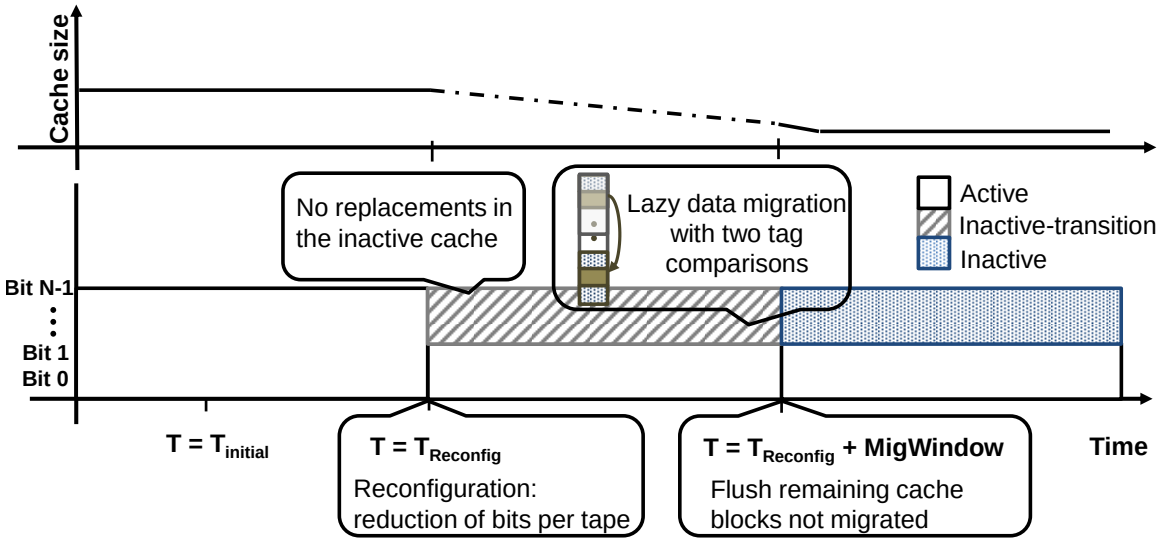


Fig. 6.5.: Timeline for reconfiguration during cache shrink

lazy shrink policy does not immediately flush the cache blocks in the inactive region after reconfiguration. Instead, the cache transitions lazily into the new reconfigured state over the *MigrationWindow* (see region marked Inactive-transition in Fig. 6.5). During the *MigrationWindow*, we perform the following actions upon each cache access: (i) We check for a hit/miss in the active region, (ii) If this results in a miss, the location in the inactive-transition region where the cache block would have resided before reconfiguration is checked. If the cache block is present in the inactive-transition region, we migrate it to the corresponding location in the active region. If the cache block is not present in either region, we declare a cache miss. At T = $T_{Reconfig}$ + $MigrationWindow$, the few remaining dirty blocks in the inactive-transition region are flushed to main memory and the entire region is invalidated. In this way, the lazy shrink policy mitigates the performance overhead associated with migrating a large number of blocks at T = $T_{Reconfig}$ at the cost of marginally higher hit latency during the *MigrationWindow*. Our experiments suggest that the number of cache blocks that needs to be flushed at T = $T_{Reconfig}$ + $MigrationWindow$ are much smaller (∼5%) compared to the total dirty blocks at T = $T_{Reconfig}$.

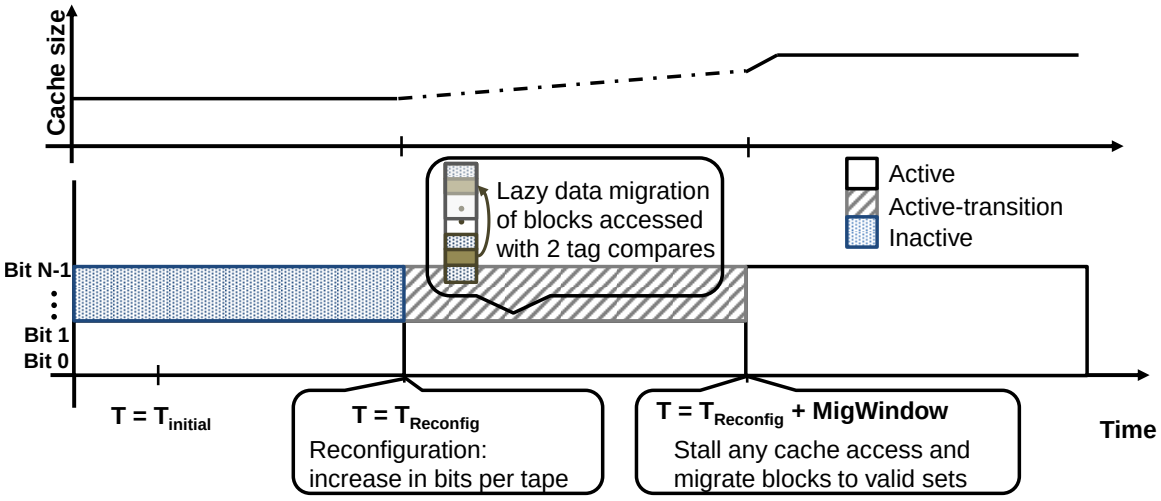


Fig. 6.6.: Timeline for reconfiguration during cache expand

Lazy expand: Fig. 6.6 illustrates the lazy expand policy for increasing the cache capacity. In this case, the newly activated portion of the cache is marked as active-transition. Subsequently, for cache accesses within the *MigrationWindow*, the blocks can be found at a location either in the active or active-transition region. Therefore, we perform two tag comparisons in the same sequence as lazy shrink, and migrate the data to its correct location (if required) during the *MigrationWindow*. At $T = T_{Reconfig} + MigrationWindow$, we stall any further cache accesses and transfer the few remaining data blocks that require migration from the active region to their valid locations in the active-transition region. In this way, the performance overheads due to data migration are significantly lowered.

Fig. 6.7 summarizes the different steps involved in a single cache access upon reconfiguration. The key point to note is that alias locations, *i.e.*, the location of cache blocks before reconfiguration, are checked only during the *MigrationWindow*. The other steps are similar to a routine cache access.

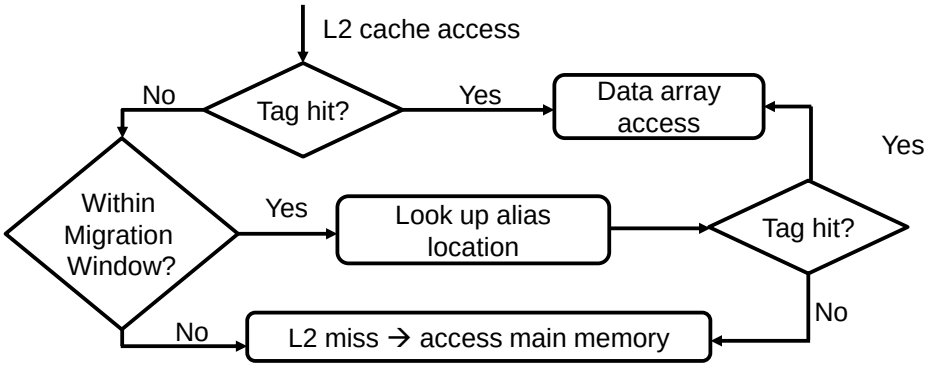


Fig. 6.7.: Flowchart for overall L2 access

6.2.4 Victim cache

When the cache is operating at less than maximum capacity, we propose to use the inactive portion as a victim cache. This victim cache holds the evicted blocks and improves the effective miss rate through reduction in conflict misses, resulting in further improvement of system performance. This modifies the sequence of steps upon a cache access as follows: (i) Upon a miss, we first check the victim cache, (ii) if we find the block, we bring it into the active cache and hold any blocks evicted during the process in the victim, else we go to main memory.

6.3 Experimental Methodology

In this section, we discuss the experimental setup used to evaluate DyReCTape. The DWM device was modeled using a self-consistent physics-based device simulation framework proposed in [59]. The device parameters were then used to obtain the read, write and shift latency (and energy) for a DWM-based cache using DWM-CACTI [8]. In our experiments, a multi-bit cell capable of storing a maximum of 64 bits with 1 read/write port is considered. We modified gem5 [97] to model the DyReCTape architecture and performed architectural simulations to evaluate it. Table 6.1 shows the baseline system configuration. The 2MB SRAM L2 cache is replaced at iso-area by DyReCTape with a maximum capacity of 64 MB. We chose a migration window of

Table 6.1.: DyReCTape system configuration

Processor Core	Alpha, out-of-order processor, 4 cores at 2 GHz
L1 I/D-cache	16KB per core, 2 way-set associative, 64B line size
L2 unified cache	2MB shared, 16 way-set associative, 64B line size
Cache latency	L1 cache: 2-cycle, L2 cache: 11-cycle
Main memory	2GB, 200-cycle

10^7 cycles with a sampling interval of 2000 accesses for dynamically reconfiguring the L2 cache. We perform a full-system simulation in the regions of interest for caches for various multi-threaded application benchmarks from PARSEC [114] and SPLASH-2x [115]. We utilize the cache access traces along with the cache characteristics from DWM-CACTI for estimating the energy consumed by DyReCTape. We use CACTI [113] for the SRAM cache, and a modified CACTI [116] for STT-MRAM in order to estimate the energy and access time. All memory technologies considered for our evaluation are based on a 32nm technology node.

6.4 Experimental Results

This section presents the results of various experiments that demonstrate the benefits of DyReCTape. with different baselines. We also demonstrate the effectiveness of our reconfiguration scheme by comparing it against: (i) the optimal capacity point for each benchmark obtained via a static sweep of cache capacities, and (ii) a naive reconfigurable cache with no lazy migration.

6.4.1 Performance evaluation

Fig. 6.8 summarizes the improvements obtained in IPC (instructions per cycle) across a wide range of benchmarks with DyReCTape. In this design, we consider six different baseline L2 cache designs under iso-area conditions: (i) an SRAM cache,

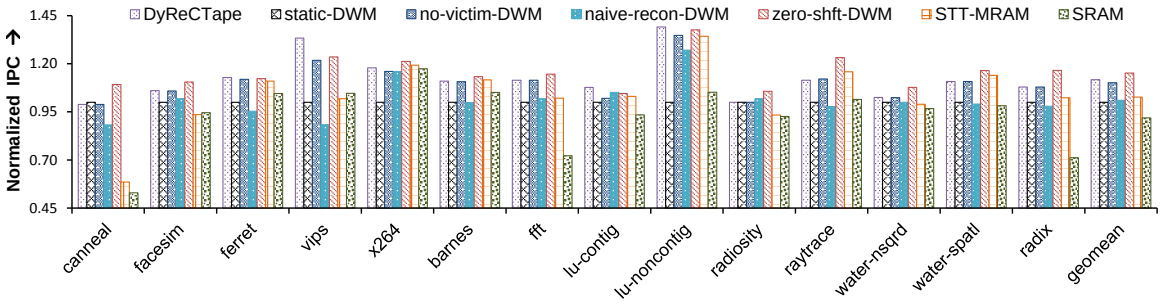


Fig. 6.8.: Performance trends for different baselines

(ii) an STT-MRAM cache, (iii) a static DWM-based cache (static-DWM) with 64MB capacity, (iv) a reconfigurable DWM cache with a naive approach in which data at invalid locations are flushed to memory or migrated to the correct location as soon as the cache is reconfigured (naive-recon-DWM), (v) a reconfigurable DWM cache with no victim cache for inactive portion (no-victim-DWM), and (vi) an "ideal" DWM cache (zero-shft-DWM) which provides the maximum capacity and also incurs no shift penalty. The IPC used for comparing these different baselines is normalized to the static organization of DWM-based L2 with maximum capacity, *i.e.*, 64MB.

On average, DyReCTape achieves 19.8% and 9% IPC improvement over the SRAM and STT-MRAM designs. This is primarily because of two factors: (a) higher cache capacity (32X and 8X *w.r.t* SRAM and STT-MRAM, respectively) due to the density of DWM, and (b) the reconfigurable cache operating at more optimal latency and capacity points for these benchmarks.

Across all benchmarks, DyReCTape obtains an 11.7% improvement (on average) in IPC over a static organization of DWM-based cache with 64MB capacity. This increase in performance is mainly because of our scheme that suitably reconfigures the cache size to operate at the capacity/latency optimal point for performance, thereby reducing the overall shift operations (3.4X on average). Fig. 6.8 also shows that a naive reconfiguration which incurs the full transition overheads hardly obtains any performance benefit ($\sim$1%). This underscores the utility of the lazy migration mechanism, which minimizes these transition overheads.

We also observe a 10.1% improvement in IPC from Fig. 6.8 over all the benchmarks, when DyReCTape uses only the proposed reconfiguration scheme without utilizing the inactive capacity as victim cache. The inclusion of victim cache provides an additional benefit of 5.8% on average for a subset of benchmarks (ferret, x264, vips, lu-contig and lu-noncontig) with substantial evictions. Note that, for some of the benchmarks in this subset (ferret, vips, lu-contig and lu-noncontig), DyReCTape outperforms even the DWM cache with no shift overheads, due to the use of a victim cache that lowers the conflict misses.

Overall, DyReCTape achieves a performance within 3.5% of the ideal DWM-based cache with no shift overheads (zero-shft-DWM), demonstrating the effectiveness of the proposed reconfiguration scheme in minimizing shift operations by utilizing the appropriate cache capacity for each application.

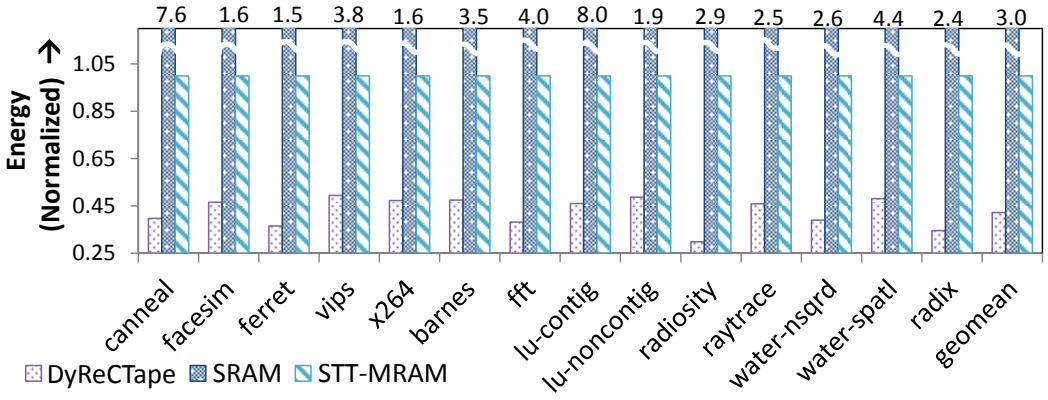


Fig. 6.9.: Energy comparison for different baselines

6.4.2 Energy comparison

Fig. 6.9 compares the energy consumed by DyReCTape with SRAM and STT-MRAM iso-area L2 caches. DyReCTape achieves a reduction of 7.1X in cache energy over SRAM. This is mainly due to the reduction in leakage energy because of the non-volatile nature of DWM. Further benefits are achieved due to techniques that lower their read/write energy [7]. In the case of STT-MRAM, we note an energy reduction

of 2.4X primarily because of lower write energy of DWMs due to the previously proposed shift-based write mechanism [7].

6.4.3 Comparison of static vs. reconfigurable DWM cache

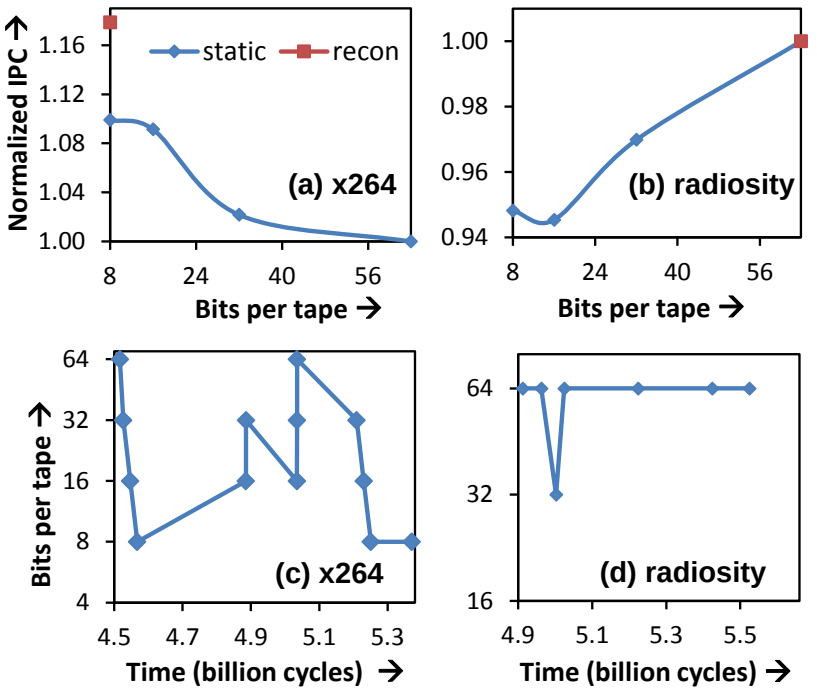


Fig. 6.10.: DyRecTape comparison against static DWM cache

In this section, we demonstrate the effectiveness of the reconfiguration scheme employed in DyRecTape against a static DWM cache of varying bits/tape. Note that, we perform comparisons at iso-area, *i.e.* higher the bits/tape, greater would be the cache capacity but with increasing shift penalty for the static case. For this analysis, we study: (i) the impact of varying the bits/tape on IPC for a static DWM cache, and (ii) the variation of bits/tape with DyRecTape over time. Fig. 6.10(a) shows a continuous decrease in performance with increasing bits/tape for a latency sensitive benchmark, x264. We also annotate the IPC achieved with DyRecTape in the figure. In this case, the reconfiguration scheme outperforms even the static best

case, mainly because of its ability to adapt to cache capacity requirements by varying the bits per tape across different program phases as shown in Fig. 6.10(c).

Fig. 6.10(b) shows a monotonically increasing trend in performance for radiosity, a capacity sensitive benchmark. In this case, the scheme achieves the same performance as the static best case. Since, this benchmark is extremely sensitive to capacity, DyRecTape does not modulate the bits/tape much and maintains the maximum possible capacity, as shown in Fig. 6.10(d).

In summary, DyRecTape either matches or exceeds the best-case static performance, demonstrating that it is an effective approach to the design of DWM-based caches.

6.5 Summary

Domain wall memory is a spintronic memory technology that provides superior density and energy efficiency compared to other emerging memory technologies. However, realizing larger capacity with such memories is challenging owing to the unique structure of DWM that requires shift operations for each access. The higher cache access latency due to shifts hurts overall system performance. In this work, we proposed DyReCTape, a DWM-based reconfigurable cache that dynamically tailors its cache size to improve performance. We propose optimizations that mitigate the overheads associated with reconfiguration. We performed architectural simulations on a wide range of benchmarks, and demonstrate substantial benefits in performance compared to a static DWM-based cache organization.

7. CONCLUSION

Computing platforms demand ever-increasing amounts of memory to keep up with increasing data sets and feed the growing numbers of cores that are enabled by each technology generation. Consequently, a large and growing portion of chip area and energy consumption is expended in memories, which face challenges with technology scaling due to increased leakage, process variations, and unreliability.

Spintronic memories that promise near-zero leakage and high density are of great interest for designing future computing systems. However, spintronic memories pose unique challenges such as high write latency and high write energy due to the fundamentally different storage and switching mechanisms that they employ. Therefore, improving the energy-efficiency and performance of spintronic memory systems remains an important challenge for designers.

7.1 Thesis Summary

To address these challenges, the thesis proposed various directions to improve the efficiency of spintronic memory subsystems. It proposed the design of approximate memory systems, wherein the accuracy of data written to or read from the memory array is relaxed for benefits in energy and performance. It also explored the design of efficient cache architectures for domain wall memories, an advanced spintronic memory technology. The key contributions of the dissertation are summarized below.

- The thesis explored approximate memory compression to reduce the memory traffic for STT-MRAM based main memory. It suggested a quality-aware memory controller design that transparently compresses/decompresses data written to/read from approximation-tolerant memory regions, while conforming to a specified error constraint for each region. A software interface was introduced

to identify the approximate-resilient memory regions, and a runtime quality control framework was proposed to automatically determine the error constraints for the identified memory regions such that a specified target application-level quality is maintained. The proposed concept was demonstrated by realizing an FPGA prototype system across a wide range of machine learning applications.

- The dissertation addressed the key challenge of energy-inefficiency for each read and write operation in spintronic memories. It identified various mechanisms at the circuit and architecture levels that trade-off energy at the cost of small read, write or retention error probabilities. Based on these mechanisms, a quality-configurable memory array was designed, with enhancements restricted solely to the peripheral circuits, while retaining the core array structure of a standard memory. The thesis proposed two different designs, STAxPad, a quality-configurable scratchpad integrated in the memory hierarchy of a programmable vector processor, and STAxCache, a quality-configurable cache architecture for a general purpose processor. The corresponding ISAs were enhanced to expose the proposed designs to software. Across a wide range of applications, the results demonstrate that quality-configurable spintronic memories can achieve considerable improvement in energy with negligible quality loss.

- Finally, the thesis explored the design of caches with a more advanced spintronic memory technology, *viz.* DWM. It presented DyReCTape, a reconfigurable cache that exploits the unparalleled density benefits of DWMs by packing the maximum number of bits within each tape while simultaneously leveraging the intrinsic capability of DWMs to modulate the active bits per tape at runtime, depending on the requirements of the applications. It also proposed two performance optimizations to DyReCTape: (i) a lazy migration policy to minimize the overheads of reconfiguration, and (ii) re-use of the unused portion of the cache due to reconfiguration, as a victim cache to reduce the number of off-chip accesses. DyReCTape was evaluated across a wide range of multi-

threaded workloads which demonstrated significant performance benefits over both SRAM and prior spintronic cache designs.

In summary, the dissertation proposes various approaches to improve the energy efficiency and performance of spintronic memories and re-affirms their potential as a replacement for state-of-the-art memory technologies.

REFERENCES

REFERENCES

[1] Y.-K. Chen, J. Chhugani, P. Dubey, C. Hughes, D. Kim, S. Kumar, V. Lee, A. Nguyen, and M. Smelyanskiy, "Convergence of recognition, mining, and synthesis workloads and its implications," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 790–807, 2008.

[2] P. Dubey, "A platform 2015 workload model recognition, mining and synthesis moves computers to the era of tera," White paper, Intel Corp., 2005.

[3] "http://www.everspin.com/."

[4] D. Apalkov, A. Khvalkovskiy, S. Watts, V. Nikitin, X. Tang, D. Lottis, K. Moon, X. Luo, E. Chen, A. Ong, A. Driskill-Smith, and M. Krounbi, "Spin-transfer Torque Magnetic Random Access Memory (STT-MRAM)," *J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 2, pp. 13:1–13:35, May 2013.

[5] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobb's journal*, vol. 30, no. 3, pp. 497 – 503, March 2005.

[6] S. Parkin, M. Hayashi, and L. Thomas, "Magnetic domain-wall racetrack memory," *Science*, vol. 320, no. 5873, pp. 190–194, Apr 2008.

[7] S. Fukami, T. Suzuki, K. Nagahara, N. Ohshima, Y. Ozaki, S. Saito, R. Nebashi, N. Sakimura, H. Honjo, K. Mori, C. Igarashi, S. Miura, N. Ishiwata, and T. Sugibayashi, "Low-current perpendicular domain wall motion cell for scalable high-speed MRAM," in *Proceedings of the IEEE Symposium on VLSI Technology*, Jun 2009, pp. 230 –231.

[8] R. Venkatesan, V. Kozhikkottu, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, "TapeCache: A high density, energy efficient cache based on domain wall memory," in *Proceedings of the International Symposium on Low Power Electronics and Design*, Jul. 2012, pp. 185–190.

[9] V. Chippa, S. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*, 2013, pp. 1–9.

[10] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate Computing and the Quest for Computing Efficiency," in *Proceedings of the 52nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, June 2015, pp. 120:1–120:6. [Online]. Available: http://doi.acm.org/10.1145/2744769.2751163

[11] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design & Test*, vol. 33, no. 1, pp. 8–22, Feb 2016.

[12] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Design exploration of hybrid caches with disparate memory technologies," *ACM Trans. Architecture and Code Optimization*, vol. 7, no. 3, pp. 15:1–15:34, Dec. 2010.

[13] A. Smith and Y. Huai, "STT-RAM - A New Spin on Universal Memory," *Future Fab Intl.*, vol. 23, July 2007.

[14] K. Lee and S. Kang, "Development of Embedded STT-MRAM for Mobile System-on-Chips," *IEEE Trans. on Magnetics*, vol. 47, no. 1, pp. 131 –136, Jan. 2011.

[15] A. Nigam, C. W. Smullen, IV, V. Mohan, E. Chen, S. Gurumurthi, and M. R. Stan, "Delivering on the promise of universal memory for Spin-Transfer Torque RAM (STT-RAM)," in *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*, Aug. 2011, pp. 121–126.

[16] R. Desikan, C. Lefurgy, S. Keckler, and D. Burger, "On-chip MRAM as a High-Bandwidth, Low-Latency Replacement for DRAM Physical Memories," In IBM Austin CASC, Tech. Rep., Sep. 2002.

[17] Y. Xie, "Modeling, architecture, and applications for emerging memory technologies," *IEEE Design and Test of Computers*, vol. 28, no. 1, pp. 44 –51, Jan-Feb 2011.

[18] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proceedings of the 36th annual international symposium on Computer architecture*, ser. ISCA '09, Jun. 2009, pp. 24–33.

[19] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: a hybrid PRAM and DRAM main memory system," in *Proceedings of the 46th Annual Design Automation Conference*, ser. DAC '09, Jun. 2009, pp. 664–469.

[20] H. Wong, S. Raoux, S. Kim, J. Liang, J. Reifenberg, B. Rajendran, M. Asheghi, and K. Goodson, "Phase Change Memory," *Proc. of the IEEE*, vol. 98, no. 12, pp. 2201 –2227, Dec. 2010.

[21] R. Venkatesan, V. K. Chippa, C. Augustine, K. Roy, and A. Raghunathan, "Domain-specific many-core computing using spin-based memory," *Nanotechnology, IEEE Transactions on*, vol. 13, no. 5, pp. 881–894, Sept 2014.

[22] A. Iyengar and S. Ghosh, "Modeling and Analysis of Domain Wall Dynamics for Robust and Low-Power Embedded Memory," in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC '14, 2014.

[23] Z. Sun, W. Wu, and H. Li, "Cross-layer Racetrack Memory Design for Ultra High Density and Low Power Consumption," in *Proceedings of the Design Automation Conference*, Jun. 2013, pp. 1–6.

[24] R. Venkatesan, M. Sharad, K. Roy, and A. Raghunathan, "DWM-TAPESTRI - An energy efficient all-spin cache using domain wall shift based writes," in *Proceedings of the Design, Automation Test in Europe*, 2013, pp. 1825–1830.

[25] R. Venkatesan, S. Ramasubramanian, S. Venkataramani, K. Roy, and A. Raghunathan, "STAG: Spintronic-Tape Architecture for GPGPU cache hierarchies," in *Proceeding of the 41st Annual International Symposium on Computer Architecture*, Jun. 2014, pp. 253–264.

[26] Z. Sun, X. Bi, A. K. Jones, and H. Li, "Design Exploration of Racetrack Lower-level Caches," in *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, 2014, pp. 263–266.

[27] N. Mojumder, S. Gupta, S. Choday, D. Nikonov, and K. Roy, "A Three-Terminal Dual-Pillar STT-MRAM for High-Performance Robust Memory Applications," *IEEE TED*, vol. 58, no. 5, pp. 1508 –1516, May 2011.

[28] N. Mojumder and K. Roy, "Switching current reduction and thermally induced delay spread compression in tilted magnetic anisotropy spin-transfer torque (STT) MRAM," *IEEE Trans. Magnetics*, 2011.

[29] C. Augustine, A. Raychowdhury, D. Somasekhar, J. Tschanz, K. Roy, and V. De, "Numerical analysis of typical STT-MTJ stacks for 1T-1R memory arrays," in *Proc. IEDM*, Dec. 2010, pp. 22.7.1 –22.7.4.

[30] J. Li, H. Liu, S. Salahuddin, and K. Roy, "Variation-tolerant Spin-Torque Transfer (STT) MRAM array for yield enhancement," in *Proc. CICC*, Sep. 2008, pp. 193 –196.

[31] Y. Kim, S. K. Gupta, S. P. Park, G. Panagopoulos, and K. Roy, "Write-optimized reliable design of STT MRAM," in *Proceedings of the 2012 International Symposium on Low Power Electronics and Design*, ser. ISLPED '12, 2012, pp. 3–8.

[32] S. Chatterjee, M. Rasquinha, S. Yalamanchili, and S. Mukhopadhyay, "A Scalable Design Methodology for Energy Minimization of STTRAM: A Circuit and Architecture Perspective," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 5, pp. 809–817, May 2011.

[33] J. Li, C. Augustine, S. Salahuddin, and K. Roy, "Modeling of failure probability and statistical design of Spin-Torque Transfer Magnetic Random Access Memory (STT MRAM) array for yield enhancement," in *Proc. DAC*, Jun. 2008, pp. 278 –283.

[34] Y. Chen and H. Li, "Emerging sensing techniques for emerging memories," in *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, jan. 2011, pp. 204 –210.

[35] Y. Chen, W. F. Wong, H. Li, and C. K. Koh, "Processor caches with multi-level spin-transfer torque RAM cells," in *Proc. ISLPED*, Aug. 2011, pp. 73–78.

[36] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *Proceedings of the International Symposium on Computer Architecture*, Jun. 2009, pp. 34–45.

[37] B. Wang, B. Wu, D. Li, X. Shen, W. Yu, Y. Jiao, and J. Vetter, "Exploring hybrid memory for GPU energy efficiency through software-hardware co-design," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, Sep. 2013, pp. 93–102.

[38] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy Reduction for STT-RAM Using Early Write Termination," in *Proceedings of the 2009 International Conference on Computer-Aided Design*, ser. ICCAD '09, 2009, pp. 264–268.

[39] S. P. Park, S. Gupta, N. Mojumder, A. Raghunathan, and K. Roy, "Future cache design using STT MRAMs for improved energy efficiency: Devices, circuits and architecture," in *Proceedings of the Design Automation Conference*, Jun. 2012, pp. 492–497.

[40] 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 Design Automation Conference*, Jun. 2012, pp. 243 –252.

[41] 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 *Proceedings of the International Symposium on High Performance Computer Architecture*, Feb. 2011, pp. 50 –61.

[42] J. Zhao, C. Xu, and Y. Xie, "Bandwidth-aware reconfigurable cache design with hybrid memory technologies," in *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, Nov 2011, pp. 48–55.

[43] Y.-T. Chen, J. Cong, H. Huang, B. Liu, C. Liu, M. Potkonjak, and G. Reinman, "Dynamically Reconfigurable Hybrid Cache: An Energy-efficient Last-level Cache Design," in *Proceedings of the Design, Automation Test in Europe*, ser. DATE '12, 2012, pp. 45–50.

[44] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A novel architecture of the 3D stacked MRAM L2 cache for CMPs," in *Proc. HPCA*, Feb. 2009, pp. 239 –249.

[45] F. Pellizzer, A. Pirovano, F. Ottogalli, M. Magistretti, M. Scaravaggi, P. Zuliani, M. Tosi, A. Benvenuti, P. Besana, S. Cadeo, T. Marangon, R. Morandi, R. Piva, A. Spandre, R. Zonca, A. Modelli, E. Varesi, T. Lowrey, A. Lacaita, G. Casagrande, P. Cappelletti, and R. Bez, "Novel u-trench phase-change memory cell for embedded and stand-alone non-volatile memory applications," in *VLSI Technology, 2004. Digest of Technical Papers. 2004 Symposium on*, Jun. 2004, pp. 18 – 19.

[46] F. Pellizzer, A. Benvenuti, B. Gleixner, Y. Kim, B. Johnson, M. Magistretti, T. Marangon, A. Pirovano, R. Bez, and G. Atwood, "A 90nm phase change memory technology for stand-alone non-volatile memory applications," in *VLSI Technology, 2006. Digest of Technical Papers. 2006 Symposium on*, 0-0 2006, pp. 122 –123.

[47] A. Pirovano, F. Pellizzer, I. Tortorelli, R. Harrigan, M. Magistretti, P. Petruzza, E. Varesi, D. Erbetta, T. Marangon, F. Bedeschi, R. Fackenthal, G. Atwood, and R. Bez, "Self-aligned u-trench phase-change memory cell architecture for 90nm technology and beyond," in *Solid State Device Research Conference, 2007. ESSDERC 2007. 37th European*, sept. 2007, pp. 222 –225.

[48] G. Servalli, "A 45nm generation phase change memory technology," in *Electron Devices Meeting (IEDM), 2009 IEEE International*, Dec. 2009, pp. 1 –4.

[49] W. Chen, C. Lee, D. Chao, Y. Chen, F. Chen, C. Chen, R. Yen, M. Chen, W. Wang, T. Hsiao, J. Yeh, S. Chiou, M. Liu, T. Wang, L. Chein, C. Huang, N. Shih, L. Tu, D. Huang, T. Yu, M. Kao, and M.-J. Tsai, "A novel cross-spacer phase change memory with ultra-small lithography independent contact area," in *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, Dec. 2007, pp. 319 –322.

[50] Y. Ha, J. Yi, H. Horii, J. Park, S. Joo, S. Park, U.-I. Chung, and J. Moon, "An edge contact type cell for phase change ram featuring very low power consumption," in *VLSI Technology, 2003. Digest of Technical Papers. 2003 Symposium on*, Jun. 2003, pp. 175 – 176.

[51] C.-F. Chen, A. Schrott, M. Lee, S. Raoux, Y. Shih, M. Breitwisch, F. Baumann, E. Lai, T. Shaw, P. Flaitz, R. Cheek, E. Joseph, S. Chen, B. Rajendran, H. Lung, and C. Lam, "Endurance Improvement of Ge2Sb2Te5-Based Phase Change Memory," in *Memory Workshop, 2009. IMW '09. IEEE International*, may 2009, pp. 1 –2.

[52] L. Jiang, Y. Zhang, and J. Yang, "Enhancing phase change memory lifetime through fine-grained current regulation and voltage upscaling," in *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, aug. 2011, pp. 127 –132.

[53] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *In International Symposium on Computer Architecture*, 2009.

[54] S. Cho and H. Lee, "Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, Dec. 2009, pp. 347 –357.

[55] G. Sun, D. Niu, J. Ouyang, and Y. Xie, "A frequent-value based PRAM memory architecture," in *Proc. ASP-DAC*, Jan. 2011, pp. 211 –216.

[56] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46, 2013, pp. 25–36.

[57] S. Motaman, A. Iyengar, and S. Ghosh, "Synergistic circuit and system design for energy-efficient and robust domain wall caches," in *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, 2014, pp. 195–200.

[58] M. Mao, W. Wen, Y. Zhang, Y. Chen, and H. H. Li, "Exploration of GPGPU Register File Architecture Using Domain-wall-shift-write Based Racetrack Memory," in *Proceedings of the Design Automation Conference*, 2014, pp. 196:1–196:6.

[59] C. Augustine, A. Raychowdhury, B. Behin-Aein, S. Srinivasan, J. Tschanz, V. K. De, and K. Roy, "Numerical analysis of domain wall propagation for dense memory arrays," in *Proceedings of the International Electron Devices Meeting*, Dec 2011, pp. 17.6.1 –17.6.4.

[60] M. Sharad, R. Venkatesan, A. Raghunathan, and K. Roy, "Multi-level magnetic RAM using domain wall shift for energy-efficient, high-density caches," in *Proceedings of the International Symposium on Low Power Electronics and Design*, Sep 2013, pp. 64–69.

[61] A. J. Annunziata, M. C. Gaidis, L. Thomas, C. W. Chien, C. C. Hung, P. Chevalier, E. J. O'Sullivan, J. P. Hummel, E. A. Joseph, Y. Zhu, T. Topuria, E. Delenia, P. M. Rice, S. S. P. Parkin, and W. J. Gallagher, "Racetrack memory cell array with integrated magnetic tunnel junction readout," in *Proceedings of the International Electron Devices Meeting*, Dec. 2011, pp. 24.3.1 –24.3.4.

[62] E. R. Lewis, D. Petit, L.O'Brien, A. Fernandez-Pacheco, J. Sampaio, A.-V. Jausovec, H.T.Zeng, D.E.Read, and R.P.Cowburn, "Fast domain wall motion in magnetic comb structures," *Nature*, vol. 9, no. 12, pp. 980–983, Dec 2010.

[63] Y. Wang and H. Yu, "An Ultralow-power Memory-based Big-data Computing Platform by Nonvolatile Domain-wall Nanowire Devices," in *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, ser. ISLPED '13, 2013, pp. 329–334.

[64] A. Jain, P. Hill, S. C. Lin, M. Khan, M. E. Haque, M. A. Laurenzano, S. Mahlke, L.Tang, and J. Mars, "Concise loads and stores: The case for an asymmetric compute-memory architecture for approximation," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–13.

[65] J. S. Miguel, J. Albericio, A. Moshovos, and N. E. Jerger, "Doppelgänger: A Cache for Approximate Computing," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO-48. New York, NY, USA: ACM, December 2015, pp. 50–61.

[66] J. S. Miguel, J. Albericio, N. E. Jerger, and A. Jaleel, "The Bunker Cache for spatio-value approximation," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.

[67] J. S. Miguel, M. Badr, and N. E. Jerger, "Load Value Approximation," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47. Washington, DC, USA: IEEE Computer Society, December 2014, pp. 127–139.

[68] M. Shoushtari, A. BanaiyanMofrad, and N. Dutt, "Exploiting Partially-Forgetful Memories for Approximate Computing," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 19–22, March 2015.

[69] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving dram refresh-power through critical data partitioning," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVI. New York, NY, USA: ACM, 2011, pp. 213–224.

[70] A. Raha, S. Sutar, H. Jayakumar, and V. Raghunathan, "Quality configurable approximate dram," *IEEE Transactions on Computers*, vol. 66, no. 7, pp. 1172–1187, July 2017.

[71] M. Jung, D. M. Mathew, C. Weis, and N. Wehn, "Efficient reliability management in SoCs - an approximate DRAM perspective," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2016, pp. 390–394.

[72] J. Lucas, M. A. Mesa, M. Andersch, and B. Juurlink, "Sparkk: Quality-Scalable Approximate Storage in DRAM," in *Memory Forum*, June 2014.

[73] K. Cho, Y. Lee, Y. H. Oh, G.-c. Hwang, and J. W. Lee, "edram-based tiered-reliability memory with applications to low-power frame buffers," in *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, ser. ISLPED '14. New York, NY, USA: ACM, 2014, pp. 333–338.

[74] I. J. Chang, D. Mohapatra, and K. Roy, "A Priority-Based 6T/8T Hybrid SRAM Architecture for Aggressive Voltage Scaling in Video Applications," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 21, no. 2, pp. 101–112, Feb 2011.

[75] D. H. Albonesi, "Selective Cache Ways: On-demand Cache Resource Allocation," in *Microarchitecture, 1999. MICRO-32. Proceedings. 32 Annual International Symposisum on*, ser. MICRO 32, 1999, pp. 248–259.

[76] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache for low energy embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 4, no. 2, pp. 363–387, May 2005.

[77] C. Lin, S. Kang, Y. Wang, K. Lee, X. Zhu, W. Chen, X. Li, W. Hsu, Y. Kao, M. Liu, W. Chen, Y. Lin, M. Nowak, N. Yu, and L. Tran, "45nm low power CMOS logic compatible embedded STT MRAM utilizing a reverse-connection 1T/1MTJ cell," in *Electron Devices Meeting (IEDM), 2009 IEEE International*, Dec 2009, pp. 1–4.

[78] C. C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari, "Bridging the processor-memory performance gap with 3D IC technology," *IEEE Design Test of Computers*, vol. 22, no. 6, pp. 556–564, Nov 2005.

[79] G. H. Loh, "3D-Stacked Memory Architectures for Multi-core Processors," in *2008 International Symposium on Computer Architecture*, June 2008, pp. 453–464.

[80] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *Hot Chips*, vol. 23, 2011.

[81] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, "Computing in memory with spin-transfer torque magnetic ram," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 470–483, March 2018.

[82] S. Jain, S. Venkataramani, and A. Raghunathan, "Approximation through logic isolation for the design of quality configurable circuits," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 612–617.

[83] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, 2013, pp. 1367–1372.

[84] A. Ranjan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan, "Approximate Storage for Energy Efficient Spintronic Memories," in *Proceedings of the 52nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, June 2015, pp. 195:1–195:6.

[85] A. Ranjan, S. Venkataramani, Z. Pajouhi, R. Venkatesan, K. Roy, and A. Raghunathan, "STAxCache: An approximate, energy efficient STT-MRAM cache," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 356–361.

[86] A. M. H. Monazzah, M. Shoushtari, S. G. Miremadi, A. M. Rahmani, and N. Dutt, "QuARK: Quality-configurable approximate STT-MRAM cache by fine-grained tuning of reliability-energy knobs," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, July 2017, pp. 1–6.

[87] S. Sardashti, A. Arelakis, P. Stenstrom, and D. A. Wood, *A Primer on Compression in the Memory Hierarchy*. Morgan & Claypool Pubs., 2015.

[88] R. B. Tremaine, T. B. Smith, M. Wazlowski, D. Har, K.-K. Mak, and S. Arramreddy, "Pinnacle: IBM MXT in a memory controller chip," *IEEE Micro*, vol. 21, no. 2, pp. 56–68, Mar 2001.

[89] J. Dusser and A. Seznec, "Decoupled Zero-compressed Memory," in *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, ser. HiPEAC '11. New York, NY, USA: ACM, 2011, pp. 77–86. [Online]. Available: http://doi.acm.org/10.1145/1944862.1944876

[90] G. Pekhimnko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Linearly compressed pages: A low-complexity, low-latency main memory compression framework," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2013, pp. 172–184.

[91] M. Ekman and P. Stenstrom, "A Robust Main-Memory Compression Scheme," in *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, ser. ISCA '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 74–85.

[92] H. Lekatsas, J. Henkel, S. Chakradhar, V. Jakkula, and M. Sankaradass, "CoCo: a hardware/software platform for rapid prototyping of code compression technique," in *Proceedings 2003. Design Automation Conference*, June 2003, pp. 306–311.

[93] H. Lekatsas and W. Wolf, "SAMC: a code compression algorithm for embedded processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 12, pp. 1689–1701, Dec 1999.

[94] L. Benini, D. Bruni, A. Macii, and E. Macii, "Hardware-assisted data compression for energy minimization in systems with embedded processors," in *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, ser. DATE, 2002, pp. 449–453.

[95] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating stt-ram as an energy-efficient main memory alternative," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2013, pp. 256–267.

[96] M. Poremba and Y. Xie, "NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories," in *2012 IEEE Computer Society Annual Symposium on VLSI*, Aug 2012, pp. 392–397.

[97] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News'11*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[98] "Micron MT41J512M8 Datasheet," https://www.micron.com/parts/dram/ddr3-sdram.

[99] "Micron EDF8164A1MA Datasheet," https://www.micron.com/products/dram/lpdram.

[100] "Everspin 256Mb DDR3 Spin-Torque MRAM (EMD3D256M08G1)," https://www.everspin.com/ddr3-dram-compatible-mram-spin-torque-technology-0.

[101] "Nios-II processor, Altera, March 2015."

[102] "Introduction to UniPHY IP, Altera, December 2013."

[103] X. Fong, Y. Kim, R. Venkatesan, S. H. Choday, A. Raghunathan, and K. Roy, "Spin-Transfer Torque Memories: Devices, Circuits, and Systems," *Proceedings of the IEEE*, vol. 104, no. 7, pp. 1449–1488, July 2016.

[104] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "ASLAN: Synthesis of Approximate Sequential Circuits," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2014, pp. 364:1–364:6.

[105] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: ACM, 2013, pp. 1–12.

[106] A. Ranjan, A. Raha, V. Raghunathan, and A. Raghunathan, "Approximate memory compression for energy-efficiency," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, July 2017, pp. 1–6.

[107] X. Fong, Y. Kim, S. Choday, and K. Roy, "Failure Mitigation Techniques for 1T-1MTJ Spin-Transfer Torque MRAM Bit-cells," *IEEE VLSI Systems'14*, vol. 22, no. 2, pp. 384–395, Feb 2014.

[108] J. Z. Sun, "Spin-current interaction with a monodomain magnetic body: A model study," *Phys. Rev. B*, vol. 62, pp. 570–578, Jul 2000.

[109] H. Naeimi, C. Augustine, A. Raychowdhury, L. Shih-Lien, and J. Tschanz, "STTRAM Scaling and Retention Failure," *Intel Technology Journal*, vol. 9, no. 2, May 2013.

[110] Y. Li, Y. Zhang, Y. Chen, and A. K. Jones, "Combating Write Penalties Using Software Dispatch for On-Chip MRAM Integration," *IEEE Embedded Systems Letters*, vol. 4, no. 4, pp. 82–85, Dec 2012.

[111] X. Fong, S. H. Choday, G. Panagopoulos, C. Augustine, and K. Roy, "Spice models for magnetic tunnel junctions based on monodomain approximation," Aug 2016. [Online]. Available: https://nanohub.org/resources/19048

[112] S. Ikeda, K. Miura, H. Yamamoto, K. Mizunuma, H. Gan, M. Endo, S. Kanai, J. Hayakawa, F. Matsukura, and H. Ohno, "A perpendicular-anisotropy CoFeB–MgO magnetic tunnel junction," *Nature materials*, vol. 9, no. 9, pp. 721–724, April 2010.

[113] "CACTI, www.hpl.hp.com/research/cacti."

[114] 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.

[115] SPLASH-2x, "http://parsec.cs.princeton.edu/doc/memo−splash2x−input.pdf."

[116] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen, "Circuit and microarchitecture evaluation of 3d stacking magnetic ram (mram) as a universal memory replacement," in *2008 45th ACM/IEEE Design Automation Conference*, June 2008, pp. 554–559.

VITA

VITA

Ashish Ranjan received the bachelor's degree in electronics engineering from the Indian Institute of Technology (BHU), Varanasi, India, in 2009. He is currently pursuing a Ph.D. degree in the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA.

Previously, Ashish was a senior member technical staff in the Design Creation Division, Mentor Graphics Corporation, from 2009 to 2012. He was a visiting researcher at Parallel Computing Lab, Intel Labs, Bangalore, India during Fall 2015. He has also interned with the Exa-scale Computing Group at Intel Corporation, Hillsboro, OR, USA during the summer of 2015. His primary research interests include architectures for emerging memory technologies, domain-specific accelerators and approximate computing.

Ashish received the university gold medal (including 11 other medals and prizes) from IIT-BHU and the Andrews Fellowship from Purdue's Graduate School. His research has received two best paper nominations from DATE 2017 and ISLPED 2014, and a best-in session award from TECHON 2016.