A Survey Of Architectural Approaches for Managing Embedded DRAM and Non-volatile On-chip Caches

Sparsh Mittal, Jeffrey S. Vetter, Dong Li

A Survey Of Architectural Approaches for Managing Embedded DRAM and Non-volatile On-chip Caches

Sparsh Mittal, *Member, IEEE,* Jeffrey S. Vetter, *Senior Member, IEEE,* and Dong Li

Abstract—Recent trends of CMOS scaling and increasing number of on-chip cores have led to a large increase in the size of on-chip caches. Since SRAM has low density and consumes large amount of leakage power, its use in designing on-chip caches has become more challenging. To address this issue, researchers are exploring the use of several emerging memory technologies, such as embedded DRAM, spin transfer torque RAM, resistive RAM, phase change RAM and domain wall memory. In this paper, we survey the architectural approaches proposed for designing memory systems and, specifically, caches with these emerging memory technologies. To highlight their similarities and differences, we present a classification of these technologies and architectural approaches based on their key characteristics. We also briefly summarize the challenges in using these technologies for architecting caches. We believe that this survey will help the readers gain insights into the emerging memory device technologies, and their potential use in designing future computing systems.

Index Terms—Review, classification, embedded DRAM (eDRAM), non-volatile memory (NVM), spin-transfer torque RAM (STT-RAM), resistive RAM (RRAM), phase change RAM (PCM), domain wall memory (DWM), emerging memory technologies.

1 Introduction

The design of memory systems is under increased scrutiny as computer architects explore solutions to managing power consumption on two fronts. First, the increasing costs of procuring and operating large scale supercomputers and data centers, have focused new attention on the large and increasing power costs of the memory subsystems [1, 2]. Also, these systems, driven by data-intensive workloads, need larger capacity memory systems, and need alternative memory technologies that can balance the costs against capacities.

Second, recent trends in CMOS scaling and processor design have led to a large increase in the number of on-chip cores. Further, to feed data to these cores and offset the limitations posed by off-chip memory bandwidth, the sizes of on-chip caches are growing. For example, Intel's Enterprise Xeon processor uses 30 MB last level cache (LLC) [3], up from just a few megabytes several years ago. Conventionally, SRAM has been used for architecting on-chip caches, due to its desirable properties such as very high write endurance, low access latency, efficient dynamic energy, and manufacturability. However, SRAM also has large leakage power consumption and low density and hence, caches designed with SRAM consume a significant fraction of chip area and power budget. For example, in the Niagara processor, L2 cache consumes nearly 25% of the total chip power budget [4]. Moreover, it has been shown [5] that to offset the memory bandwidth limitations, large size caches need to be used and if such caches are designed with SRAM, they may occupy 90% of the chip-area in future CMOS generations. Although architectural and system-level efforts may partially address some of these challenges [4], the performance-targets of modern processor design present much higher levels of demand than what state-of-the-art SRAM-based designs may fulfill.

Due to their several desired features, emerging memory technologies such as eDRAM, STT-RAM, RRAM, PCM and DWM[1] present as promising technologies for solving some of these challenges. When used for designing on-chip caches, these technologies have much higher density and lower leakage than SRAM, which reduces the area and power consumption of the cache for same capacity or increases the cache capacity for the same area. With ongoing research work, these technologies are maturing from prototype stage to product stage. Currently, several commercial vendors have released these chips in the market [20–22]. Also, these technologies are already appearing in high-performance processors, for example, the last level caches in IBM's Power 7 processor [23] and Blue Gene/L supercomputer chip [24] are de-

• *The authors are with the Future Technologies Group at Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA 37830.*
E-mail: {mittals,vetter,lid1}@ornl.gov

1. Other commonly used acronyms for these memory technologies are: PCRAM (phase change RAM) or PRAM, ReRAM (resistive RAM or memristor), STT-MRAM (spin-transfer torque magnetic RAM). Also, spin torque transfer RAM is also sometimes presented as the expansion of the acronym STT-RAM (e.g. [19]). DWM is also referred to as racetrack memory.

TABLE 1: Characteristics of Different Memory Technologies [6–18] (R/W = read/write)

	SRAM	EDRAM	STT-RAM	RRAM	PCM	DWM
Cell size (F^2)	$120-200$	$60-100$	$6-50$	$4-10$	$4-12$	≥ 2
Write Endurance	10^{16}	10^{16}	4×10^{12}	10^{11}	$10^{8}-10^{9}$	10^{16}
Speed (R/W)	Very fast	Fast	Fast/slow	Fast/slow	Slow/very slow	Fast/slow
Leakage Power	High	Medium	Low	Low	Low	Low
Dynamic Energy (R/W)	Low	Medium	Low/high	Low/high	Medium/high	Low/high
Retention Period	N/A	$30-100$ μs	N/A (unless relaxed)	N/A	N/A	N/A

signed using eDRAM. Similarly, Intel's Haswell processor uses 128MB eDRAM L4 cache [25]. However, these technologies are not strictly superior to SRAM on all parameters. For instance, eDRAM has a retention period (the duration for which data are preserved in the device in the absence of an external power source) in range of tens of microseconds and hence, requires refresh and the other non-volatile memories have high write latency and energy. Thus, utilizing these emerging technologies at the architecture level such that they may augment or, perhaps, even replace SRAM is a major research challenge.

In this paper, we survey several architectural techniques which have been proposed to address this challenge. We cover embedded DRAM along with four non-volatile memory technologies (STT-RAM, RRAM, PCM, DWM) and collectively refer to them as emerging memory technologies. We first provide a brief background on the physical properties of each memory technology and then present a comparative evaluation of their features and limitations. To help the readers gain insights into the similarities and differences of the techniques, we present a classification of the techniques on various parameters and also discuss the common architectural ideas used in different techniques. We classify the works based on the memory technologies used, application domain, levels of cache hierarchy, optimization goal and essential approach. The aim of this paper is to equip computer architects with the knowledge of state-of-the-art in cache design using emerging memory technologies and motivate them to develop even better techniques for the computing systems of tomorrow.

Since it is infeasible to fully discuss the research area of emerging memory technologies in a work of this length, we restrict the scope of this paper in the following way. We only discuss the use of these technologies for designing on-chip caches, although they are also used for designing main memory and register files. We only discuss architecture and system level techniques and not circuit-design level approaches, although they may also help in addressing the issues related with these technologies. Also, given the amount of research done in topics such as multi-bit storage in NVM (non-volatile memory), 3D stacking and reliability, doing full justice to these topics would require a separate research article and hence, we only mention these topics briefly. Since different techniques have been evaluated using different experimentation platforms and workloads, we do not show their qualitative results, instead, we focus on the key idea of each technique.

The rest of the paper is organized as follows. Section 2 presents a comparative evaluation of these technologies and also highlights their features and research challenges. Section 3 presents a classification and overview of the techniques. Sections 4 and 5 discuss some of the techniques in detail. Finally, section 6 presents the conclusion.

2 A Comparative Evaluation of Emerging Memory Technologies:

In this section, we provide a brief background of the memory technologies and also discuss their features and the challenges in their use. Table 1 presents a comparative evaluation of the properties of different technologies. Here F denotes the smallest lithographic dimension in a given technology node. Also note that these values may not be the best-in-class, since continuous research may lead to significant improvement in their properties.

2.1 Embedded DRAM

Embedded DRAM is a capacitor-based dynamic RAM that can be integrated on the same die as the processor. It can be either conventional 1T1C DRAM cell or a logic compatible gain cell [11, 26], both of which use some form of capacitor to store the data. Since the stored charge gradually leaks away, periodic refresh is required to preserve its value and prevent decay. A typical multi-megabyte (e.g. 4MB) eDRAM cache has read/write latency in the range of 3-5 nanoseconds [11, 27].

Requirement of periodic refresh is a major obstacle in the use of eDRAM. Embedded DRAM uses fast logic transistors which have higher leakage than the conventional DRAM and hence, its refresh requirement is also higher than that of the DRAM. Barth, et al. [28] report the retention period of eDRAM to be 40μs, in comparison to the 64ms retention period of a commodity DRAM [29]. Also, the retention period reduces further due to factors such as temperature increase, process variation and technology scaling [11, 30]. Refresh operations consume a significant fraction of cache energy, interfere with cache access and reduce cache availability. Also, for small retention

periods and large sized caches, a very large number of blocks need to be refreshed in a small amount of time. Finally, eDRAM is considered to have scalability challenges due to the difficulty of precise charge placement and data sensing.

2.2 STT-RAM

STT-RAM [13] utilizes a Magnetic Tunnel Junction (MTJ) as the memory storage. An MTJ contains two ferromagnetic layers separated by an oxide barrier layer. The magnetization direction of one ferromagnetic layer is fixed while that of the other ferromagnetic layer can be altered by passing a current. The resistance of the MTJ is determined by the relative magnetization direction of these two layers. If the two layers have different directions, the resistance of the MTJ is high and vice versa. Using this property, a binary value is stored in an STT-RAM cell.

Although STT-RAM has lower density than PCM and RRAM and higher write latency and energy than SRAM, it has been widely used for designing caches due to its high write endurance. However, note that for STT-RAM, although a write endurance value of 10^{15} has been estimated, the best endurance test result so far is less than 4×10^{12} writes [31]. Another advantage of STT-RAM is that its non-volatility can be traded to improve its write energy and latency. Smullen et al. [13] relax the retention time by shrinking the MTJ planar area, while Jog et al. [32] achieve this by decreasing the thickness of the free layer and lowering the saturation magnetization which reduces the thermal barrier of MTJ. As an example, Jog et al. [32] show that for 2GHz frequency, the write latency values of a 4MB STT-RAM for retention periods of 10 years, 1 second and 10 milli-second are 22, 12 and 6 cycles, respectively. Thus, based on the application characteristic and the level of cache hierarchy, a designer can choose a suitable value of retention period.

2.3 RRAM

An RRAM with unipolar switching uses an insulating dielectric [33]. When a sufficiently high voltage is applied, a filament or conducting path is formed in the insulating dielectric. After this, by applying suitable voltages, the filament may be set (which leads to a low resistance) or reset (which leads to a high resistance).

Compared to SRAM, an RRAM cache has high density, comparable read latency, and much smaller leakage energy. However, the limitation of RRAM is its low write endurance of 10^{11} [34] and high write latency and write energy. For example, a typical 4MB RRAM cache has a read latency of 6-8 nanoseconds and a write latency of 20-30 nanoseconds [35].

2.4 PCM

PCM uses a phase change material called GST, which is an alloy of germanium, antimony, and tellurium. When the alloy is heated a very high temperature and quickly cooled down, it transitions into an amorphous substance with high electrical resistance. On the other hand, when the alloy is heated to a temperature between the crystallization and melting point and cooled down slowly, it crystallizes to a physical state with lower resistance. This physical property is used to store a binary value in a PCM cell.

The two most severe challenges in using PCM for designing on-chip caches are its limited write endurance and high write latency. Since the write traffic to a cache is much heavier than that to a main memory, and the write endurance of PCM is only near 10^8 writes [36, 37], for several applications, a PCM cache can fail in less than an hour. A typical 4MB PCM cache has a read latency of 15-20 nanoseconds and a write latency of 150-170 nanoseconds [35, 37]. Thus, PCM is suitable for main memory or lower-level cache hierarchies (e.g. L3 cache or even L4 cache [10]) where its high latency can be tolerated and the high-density can be leveraged to avoid off-chip accesses.

2.5 DWM

DWM works by controlling domain wall (DW) motion in ferromagnetic nanowires [18]. The ferromagnetic wire can have multiple domains which are separated by domain walls. These domains can be individually programmed to store a single bit (in the form of a magnetization direction) and thus, DWM can store multiple bits per memory cell.

Logically, a DWM macro-cell appears as a tape, which stores multiple bits and can be shifted in either direction. The challenge in using DWM is that the time consumed in accessing a bit depends on its location relative to the access port, which leads to non-uniform access latency and makes the performance dependent on the number of shift operations required per access. Compared to other NVMs, DWM is less mature, and is still in research and prototype phase.

2.6 Characteristics common to NVMs

Unlike charge-based memories such as DRAM, NVMs store data in the form of change in physical state. Since a write to NVM involves changing its physical state, a write operation to NVM consumes larger time and energy than a read operation, leading to read-write asymmetry [38]. Similarly, the write-latency and energy of logical $1 \rightarrow 0$ transition is higher than that of $0 \rightarrow 1$, leading to $0/1$ write asymmetry [39]. NVMs also allow storing multiple data-bits in single memory cell. This is referred to as MLC (multi-level cell) storage and it leads to a significant increase in the storage density of NVMs (see Table 2).

3 Classification and Overview

Table 2 presents a classification and overview of different approaches. It presents the classification of the existing works on several dimensions, such as the memory technologies (e.g. STT-RAM, eDRAM etc.) and hybrid approaches (at both same and different levels of cache). Further, it classifies them based on the application domain (CPU/GPU or L1/L2) since the the design-requirement of each domain is different. Table 2 further classifies the works based on the goal of the cache management technique and the essential approach, to allow the reader to see both similarities and differences between different works. This classification is expected to be useful for researchers, CAD-designers and technical marketing-professionals. While a technique proposed for improving performance may also lead to saving in energy, and a technique designed for one NVM may also work equally well for other NVMs, in this classification, we include a technique in a place for which the technique has been actually evaluated.

We now discuss the basics and essential ideas of architecture-level cache management which are used in different techniques discussed in the paper.

Leveraging temporal locality: Caches work on the temporal locality principle, which states that a data item which is referenced at one point of time will be referenced again in near future. Such data item is stored in caches to avoid accessing main memory. However, when the program phase changes, the existing data items in the cache will no longer be reused. Such blocks are referred to as dead-blocks and they constitute a large fraction of total cache blocks [4, 16, 26, 110]. Before getting evicted, they waste leakage energy and refresh energy (in eDRAM caches) [11]. By detecting them early and evicting them, the energy effiiency can be improved. Also, in hybrid caches, early eviction of dead blocks from SRAM improves its utilization [55]. The dynamic cache reconfiguration based techniques [12, 14] work on the principle that there exists large variation in intra-application and inter-application cache requirement of different programs. Thus, by only allocating suitable amount of cache to an application in each phase, significant amount of energy can be saved.

Utilizing specific properties of each level of cache: Modern processors use a hierarchy of caches, where the cache closest to the processor is called first-level or upper-level cache, while that closest to main memory is called last-level or lowest-level cache. The optimization target for designing each level of cache is different [4]. The first-level cache is accessed very frequently and hence, it should have high speed and write endurance, even if it has a small size or high leakage power. The last level cache is designed to reduce off-chip accesses and hence, it must have large capacity. A high latency of last level cache can be usually tolerated using techniques such as out-of-order execution. For this reason, most of the research works assume or recommend that L1 cache be designed using SRAM for performance reasons [97]. If an NVM, such as STT-RAM is used for designing L1 cache, it is designed for low write latency which requires

TABLE 2: Classification of Approaches

Classification	References
Memory Technology	
EDRAM	[10, 11, 14, 26, 29, 30, 40–48]
STTRAM	[7, 8, 10–15, 19, 32, 38, 49–95]
RRAM	[36, 94, 96–98]
PCM	[37, 64, 99, 100]
DWM	[18, 91, 101, 102]
Hybrid (Different technologies at the same cache level)	
SRAM + eDRAM	[10, 40, 41, 43, 46]
SRAM + STTRAM	[7, 8, 10, 12, 54, 55, 57–59, 61, 65, 67–69, 82, 84, 89, 92, 103]
STTRAM + DRAM	[104]
STTRAM + PCM	[64]
SRAM + PCM	[10, 100]
SRAM + DWM	[18]
Hybrid (Different technologies at different cache levels)	
SRAM (L2) + eDRAM or STT-RAM (L3) + PCM (L4)	[10]
SRAM (L2) + STT-RAM (L3) + eDRAM (L4)	[14]
Application Domain	
GPU caches	[92, 105, 106]
CPU caches	almost all others
Level in cache hierarchy	
First-level cache	[13, 41, 54, 60, 68, 83, 87, 91, 93, 99, 107]
Middle or last level cache	almost all others
Goal of the cache management technique	
Energy saving	[7, 8, 10–12, 14, 15, 18, 19, 26, 30, 32, 37, 40–43, 45, 47, 48, 50, 51, 53–55, 57, 58, 61, 65, 72, 80, 81, 83, 85, 87, 89–91, 95, 97, 99, 101–103, 107]
Performance improvement	[7, 15, 32, 47, 48, 51, 53, 54, 57, 58, 61, 66, 72, 80, 85, 88, 91, 102]
Improving lifetime	[8, 36, 37, 56, 82, 86, 96, 98, 100]
Mitigating cache obstruction	[51]
Optimizing bandwidth provided by the cache hierarchy	[14]
Providing quality-of-service	[84]
Essential Approach/Feature	
Minimizing or avoiding writes	[19, 37, 38, 50, 51, 53, 55, 56, 58, 61, 69, 80, 81, 89, 100]
Data migration b/w SRAM and NVM/eDRAM in hybrid caches	[7, 8, 10, 40, 41, 43, 46, 55, 58, 59, 61, 67, 69, 82, 103]
Cache reconfiguration	[12, 14, 45, 47, 87]
Reducing refresh operations	[11, 30, 42, 44, 44, 45, 47, 48, 72]
Using volatile STT-RAM	[11, 13, 32, 54, 60, 72, 78]
Addressing asymmetric 1/0 writes in NVMs	[39, 88]
Additional cache levels/buffers	[19, 50, 61, 68, 87, 107]
Cache color-level wear-leveling Set-level wear-leveling Way-level wear-leveling Memory cell-level wear-leveling	[86] [36, 79, 82] [36, 57, 82, 96, 98] [37, 56]
Cache partitioning	[8, 84]
Using compiler	[7, 58, 59, 67, 69, 92, 93]
Using MLC NVM	[18, 49, 79, 91, 108]
3D-stacking of NVM	[7, 10, 61, 63, 70, 84, 105, 109]

relaxing its retention time [13, 60, 107]. In general, based on their latency and write endurance values, along with capacity advantage, emerging memory technologies discussed here are more suitable for the last level cache than the first level cache.

Using wear-leveling for lifetime enhancement: Depending on the program memory access behavior and cache replacement policy, a large intra-set and inter-set variation may exist in the number of writes to the cache blocks [111]. Since SRAM has very high write endurance, this phenomenon does not cause problems in the SRAM caches and hence, conventional cache management policies are write-variation unaware. NVM caches, however, have several orders of magnitude smaller write endurance. Hence, high write-variation may cause some memory cells to fail much early than the remaining cells, which may shorten the cache lifetime. To address this, wear-leveling techniques have been proposed [36, 37, 82, 96] which aim to evenly distribute the writes to different cache blocks. Note that, the maximum improvement in lifetime which can be achieved through wear-leveling is limited by the write-variation originally present in the application.

Reducing number of writes to NVMs: Another approach to mitigate the problem of limited write endurance is to reduce the number of writes at the bit-level or cache-access level. At bit-level, the redundant writes can be removed, i.e. those requests which write the same bit to the block which is originally written [37, 80]. This can be achieved through a read-before-write operation, which is advantageous since for NVMs, a read operation incurs much less overhead than a write operation. Also, writing of some frequent patterns such as all-zero bits can be avoided by using flags [53]. Further, by using special data-encoding scheme, bit-switching probability can be reduced [56]. At cache-access level, the number of writes can be reduced by using buffers or another level of cache [19, 50, 61], which coalesce write accesses. Also, by minimizing the eviction of dirty data in upper level of cache [19] or avoiding writing the unchanged sub-blocks [81], or using cache bypassing [51], the number of writes can be reduced. Note that reduction in number of writes also improves performance and energy efficiency; and some of the above techniques can be synergistically combined as they are orthogonal to each other.

Using hybrid caches: A cache has a set-associative structure, where a newly-arrived cache block[2] may reside in any of the cache ways, as decided by the cache replacement policy. Physically, these cache ways may be designed using either the same technology (e.g. all STT-RAM ways) or different technologies (e.g. some ways are designed using SRAM, while others are designed using STT-RAM) or same technology but different properties (e.g. retention and response time [60, 78]). Several hybrid cache designs use this approach to architect different cache ways using disparate memory technologies (e.g. [41, 43, 103]). In this way, the best feature of each technology is leveraged, while overcoming its limitations through the use of other technology. For example, the low-density of SRAM can be overcome by using high-density NVMs while leveraging its fast speed. At the same time, the limited write endurance of NVMs can be overcome by using SRAM.

Using 3D stacking: 3D die stacking increases transistor density by vertically integrating multiple dies with a high-bandwidth, high-speed and low-power interface. Using 3D die stacking, dies of even different types can be stacked. Several researchers discuss 3D stacking of NVM caches (e.g. [7, 10, 61, 70, 84]). As an example, Sun et al. [61] propose fabricating STT-RAM caches and CMP logic as two separate dies and then stack them together in a vertical manner. One benefit of this is that the magnetic-related fabrication process of STT-RAM may not affect the normal CMOS logic fabrication. 3D stacking also enables shorter global interconnect, lower interconnect power consumption and smaller footprint.

4 Cache Management Approaches

We now discuss some of the techniques in detail.

4.1 Approaches Using eDRAM

4.1.1 Using error-correcting codes

Wilkerson et al. [29] propose using strong (multi-bit) error-correcting codes which allow tolerating higher number of errors. This allows increasing the refresh period and thus, the requirement for refresh can be reduced. The limitation of this approach is that use of strong codes incurs overheads in terms of storage, encoding/decoding power, and area.

4.1.2 Minimizing refresh operations

Reohr [44] discusses several approaches for refreshing eDRAM caches, for example, simple periodic refresh and no-refresh etc. In periodic refresh, all lines at refreshed periodically before their retention period. The 'no refresh' policy maintains a counter with each cache line, which starts counting after any cache access to that line. If the next access to that line does not occur within a refresh period, the line is marked as invalid and the data, if dirty, are written back to memory. This policy does not incur any refresh overhead, but it significantly increases the number of cache misses.

Agrawal et al. [26] propose schemes to reduce the refresh operations in an eDRAM cache by utilizing the fact that on a read or a write, a cache line is

2. In this paper, we use the terms 'cache line' and 'cache block' synonymously.

automatically refreshed and hence, it need not be refreshed for the duration of one retention period. One of their schemes, named 'polyphase', divides the retention period into multiple (e.g. 2, 4 etc.) phases. Each cache line maintains the information about the phase in which it was last accessed. Afterwards, instead of refreshing the cache line at the beginning of next refresh period, their technique refreshes the line at the beginning of this phase in the next refresh period. Another scheme avoids refreshing invalid blocks, while a third scheme refreshes only dirty blocks, since decay of clean blocks in absence of refresh does not cause data inconsistency and these blocks can be later restored from the main memory. A fourth scheme, named WB(n,m) refreshes idle dirty lines n times before writeback and idle valid clean lines m times before invalidation. The motivation behind this is to avoid refreshing the dead lines, while also keeping the number of extra misses small.

Chang et al. [11] propose a technique for reducing eDRAM refresh power by avoiding the refresh operations for the dead blocks. To determine such blocks, they use a deadline prediction scheme, where a line is predicted to be dead if it has not been accessed for a certain time duration.

Mittal [45] proposes a technique to save both leakage and refresh energy in eDRAM caches. His technique uses dynamic cache reconfiguration approach to turn-off part of the cache to save leakage energy and refreshes only valid data of the active (i.e. not turned-off) cache to save refresh energy. Thus, due to reduction in the active region of the cache, the refresh requirement is also reduced. His technique uses cache coloring scheme [16, 112] to achieve cache reconfiguration, along with set-sampling based profiling to obtain dynamic profiling data. While the above mentioned techniques [11, 26, 44] avoid refresh operations at cache-line granularity, the technique by Mittal [45] avoids refresh operations first at cache-color granularity (using cache reconfiguration) and then at cache-line granularity. The limitation of this technique is that due to the use of cache coloring, a change in cache reconfiguration requires flushing the cache, which may increase the accesses to main memory.

Alizadeh et al. [30] propose a technique for effectively scheduling refresh operations in multi-banked eDRAM caches. They decouple the refresh scheduler into two modules, one of which determines which cell to refresh next and the other module determines when to force an idle cycle in all banks. Utilizing this support, their technique performs concurrent refresh, i.e. an idle bank is refreshed when a read/write operation is being performed in other banks.

4.1.3 Accounting for variation in retention time

Agrawal et al. [48] build a mathematical model of eDRAM retention time variation and note that retention times of cells in large eDRAM caches exhibit spatial correlation. Based on this, they logically divide an eDRAM cache into logical regions (called tiles), profile the retention characteristics of each tile, and then program their refresh requirements in the cache controller. Using this, each tile can be refreshed at a different rate, which significantly minimizes the number of refresh operations.

4.2 Approaches Using STT-RAM

4.2.1 Relaxing non-volatility of STT-RAM

The ability to trade-off STT-RAM retention period with performance provides the designers significant flexibility. STT-RAM caches with different retention periods have been used both at the same level [60] and at different levels [13, 60, 107] of cache hierarchy. We now discuss some of these techniques.

Smullen et al. [13] propose relaxing STT-RAM retention time to improve its performance. For a three level cache hierarchy, they experiment with two cases; first, in which all the three levels of cache are designed using STT-RAM and second, in which the L1 cache is designed using SRAM while the last two levels of cache are designed using STT-RAM. They observe that the although the first case provides the highest energy efficiency, its performance is lower since the access frequency of L1 is high and even with a write-optimized design, the write latency of STT-RAM is higher than that of SRAM. The second case provides performance level close to the pure-SRAM baseline design, while significantly improving the energy efficiency over the baseline.

Jog et al. [32] propose an STT-RAM L2 cache design where a suitable value of retention period is chosen based on the applications' inter-write times, such that the refresh overhead is minimized and the performance is improved. They observe that a retention period of 10ms is suitable for L2 cache. For those applications which have inter-write times higher than this value, refresh operation is performed for diminishing blocks (i.e. those blocks which are about to lose their data) by temporarily storing them in a buffer.

Guo et al. [107] propose the use of STT-RAM to design combinational logic, register files and on-chip storage (I/D L1 caches, TLBs and L2 cache). They evaluate multiple cache designs where the STT-RAM cache (both L1 and L2) can have either the same capacity as the SRAM cache (to reduce area and interconnect delays) or the same area as the SRAM cache (to increase capacity and reduce misses). The STT-RAM cells used for L1 are optimized for write speed while those used for L2 are optimized for density and access energy. Also, to hide the write latency of STT-RAM, they propose subbank buffering which allows the writes to complete locally within each subbank, while the reads from other locations within the

array can complete unobstructed. They show that by carefully designing the pipeline, the STT-RAM based design can significantly reduce the leakage power, while also maintaining the performance level close to the CMOS design.

Sun et al. [60] propose an STT-RAM cache design for lower level caches where different cache ways are designed with different retention periods. For example, in a 16-way cache, way 0 is designed with a fast STT-RAM design with low retention period and the remaining 15 ways are designed with a slow STT-RAM design which has higher retention period. Their technique uses hardware to detect whether a block is read or write intensive. The write intensive blocks are primarily allocated from way 0, while the read intensive blocks are allocated from the other ways. Also, to avoid refreshing dying blocks in way 0, their technique uses data migration to move such blocks to banks with higher retention period.

For designing L1 cache, Sun et al. [60] propose using an STT-RAM cell which is optimized for write performance and energy. Since this cell has a retention period in range of tens of microseconds, a refresh mechanism is required. For this, they propose using counters with each cache block. Using this, refresh is performed only on those cache blocks that have reached their full lifespan.

4.2.2 Minimizing or avoiding writes

Early write termination techniques may leverage either read-write asymmetry [37, 38, 80, 92], or 0/1 write asymmetry [39, 88]. Also, redundant writes can be avoided by either comparison of actual data [37, 80] or by using additional flags [53, 81] which indicate certain data patterns. The discussion of a few techniques follows.

Zhou et al. [80] propose a technique to minimize the number of writes to an STT-RAM cache by avoiding bit-writes which are redundant, i.e. which write the same value in a bit, that is already stored in the cache. For this, instead of reading the original value before writing the new, they utilize the fact that when a write operation is performed on an STT-RAM cell, the resistance of MTJ does not gradually. Instead, it changes abruptly at the end of a write operation. Thus, at the early stage of a write operation, an STT-RAM cell still holds its valid value. By sampling this value at the early stage of write operation, the write current can be throttled if the old value is same as the new value.

Yazdanshenas et al. [56] propose a technique for reducing switching probability in cache by reducing the probability of memory cells being in logical "1" (high) state and then applying early write termination [80] at circuit level to eliminate redundant writes. They use limited weight codes for encoding frequent data patterns, which greatly increases the opportunity of avoiding redundant writes, since the Hamming distance between these codes is small (e.g. 2). Their data encoding scheme also has the advantage that it results in a uniform wear-out which leads to improvement in lifetime of the cache.

Ahn et al. [50] propose a technique which can be used with read-before-write schemes to further reduce the write activity to an STT-RAM cache. Their technique works on the observation that in many applications, computed values are varied within small ranges and hence, the upper bits of data are not changed as frequently as the lower bits. Based on this, they use a small SRAM cache (called lower-bit cache or LBC) between the L1 cache and the STT-RAM L2 cache. LBC holds lower half of every word in cache blocks written by the L1 data cache. For an L2 read operation, the LBC is accessed in parallel to the L2 cache. When there is a hit in LBC, the data to be provided for the L1 cache is a combination of upper bits from the L2 cache and lower bits from the LBC, since the lower bits in L2 may not be up-to-date. Similarly, when an L2 block is evicted, an up-to-date value of lower-bits in retrieved from the LBC and after that, the data in LBC is also invalidated. LBC helps in coalescing several write-requests and also hides frequent value changes from the L2 cache so that the read-before-write strategy can more effectively cancel the write requests.

Sun et al. [61] propose a write-buffer design to address the long write latency of last level (L2) STT-RAM cache. The L2 may receive a request from both L1 and write buffer. Since read latency of STT-RAM is smaller than the write latency and also reads are performance-critical, the buffer uses a read-preemptive management policy, which ensures that a read request receives higher priority than a write request. They also propose a hybrid SRAM and STT-RAM cache design which aims to move the most write-intensive blocks to SRAM. In the cache, one cache way is implemented using SRAM and the remaining ways are implemented using STT-RAM. On a write miss, the cache controller tries to first place the data in the SRAM way. To increase hits to the SRAM ways, the cache controller migrates data from STT-RAM to SRAM if there are two successive write operations to the data. They also show that combining the hybrid cache with read-preemptive write-buffer leads to additional energy saving and performance improvement. The limitation of this approach is that transfer of data between write-buffer and L2 cache causes performance overhead.

Rasquinha et al. [19] propose two techniques viz. use of a write cache and write-biasing to reduce the number of writes to a last level (L2) STT-RAM cache and also save energy. The first technique adds a small cache, called write-cache (WC) between L1 and L2. WC is a fully-associative cache which stores only dirty lines evicted from L1 and is mutually exclusive with

L2. On a cache access, both L2 and WC are accessed in parallel. The store misses are allocated in WC and the load misses are allocated in L2. WC reduces the number of L2 writes by absorbing most of the L1 writebacks. The write-biasing technique reduces the number of writebacks from L1 to lower level caches by biasing dirty cache lines to reside in L1 for a longer time. For this, the cache replacement policy uses different insertion and promotion policies for loads and stores.

Jung et al. [53] propose a technique for minimizing the writes to STT-RAM caches. Their technique works on the observation that on average, a large fraction (e.g. more 50%) of bytes and words (1 word = 4 bytes) written to the L2 cache comprise of only zero-valued data. Based on this, their technique adds additional "all-zero-data" flags in the tag arrays at the granularity of a single byte and a single word. Before any cache write, the data value is checked. If the all-zero bytes or words are detected, the corresponding flags are set and only the non-zero bytes or words are written. During a cache read operation, only the non-zero bytes or words are read and then the actual data are constructed by combining the information from the all-zero flags.

Park et al. [81] propose a technique to reduce the write activity on a last level STT-RAM cache. They logically divide the cache line into multiple partial lines, e.g. a 64B cache line is divided into 4 partial lines. In L1 cache, a history bit is kept for each partial line to track which partial lines have changed. Using this information, when a dirty L1 block is written to last level cache, only those partial lines are written which have been changed. Also, to reduce the read energy, they provision sequential tag-data access to an STT-RAM cache. On a cache read access, initially only the tag values are read and if there is a hit, only the selected cache line is accessed which reduces the dynamic energy.

4.2.3 Addressing high write latency of STT-RAM

Since high write latency of STT-RAM can lead to harmful consequences, several techniques have been proposed to address this. We now discuss some of these techniques.

Since STT-RAM has high write latency, aggressive prefetching may lead to bank-conflicts and cause performance degradation. Mao et al. [66] propose techniques for mitigating the write pressure caused due to prefetching in STT-RAM based LLC. One of their techniques prioritizes different types of LLC requests such as load, store, prefetch, and write back etc. based on their criticality. The critical requests are assigned a high priority and hence, they are served earlier. In multicore systems, the excessive requests generated from a cache-intensive program may block those generated from a cache-unintensive program which may lead to its starvation. To address this, they propose another technique which prioritizes the requests from a cache-unintensive program, so that they are served promptly. Since these two techniques are orthogonal, they can be combined to provide even better performance.

Wang et al. [51] propose a technique to address the issue of cache obstruction in STT-RAM LLCs. In a single-port STT-RAM cache, an ongoing write can cause port obstruction and delay the subsequent performance-critical read operations. Also, in a multi-core system, write requests from one core can block all the read requests to the same cache bank from other cores, resulting in performance degradation. This phenomenon is termed as cache obstruction. To identify the applications which may cause this, their technique monitors whether an application generates intensive write requests. For such applications, LLC is bypassed and main memory is directly accessed. This avoids contention and leads to improved performance.

Dong et al. [15] evaluate the effectiveness of STT-RAM for designing L2 caches as a replacement for SRAM or DRAM. For a similar area, STT-RAM and DRAM have higher capacity, while SRAM has shorter access latency. Due to the interaction of these factors, STT-RAM provides matching performance to SRAM, while DRAM provides lower performance due to its high access latency. They also evaluate use of STT-RAM for designing L3 cache, which is advantageous since it provides much lower latency than accessing main memory. They observe that especially for memory-intensive benchmarks, use of an STT-RAM L3 provides large performance improvement.

4.2.4 Addressing 0/1 write-asymmetry

Kwon et al. [88] propose a technique to reduce the write latency of STT-RAM cache. In an STT-RAM cell, the time taken in $1 \rightarrow 0$ transition is more than that taken in $0 \rightarrow 1$ transition, due to the asymmetry in the switching times of the MTJ. Thus, the time taken in a write operation is limited by the slower transition. Hence, if all the cells in a block are preset to 0, then the $1 \rightarrow 0$ transition is not required and the effective write latency can be reduced. Based on this idea, their technique introduces multiple redundant blocks (RBL) in each row of the cache data array. At the time of a write operation, the RBL is checked and if it is preset to zero, instead of writing the data to the actual data block (DBL), the data are written to RBL and the RBL is marked as the data block and the original data block is now marked as the RBL, thus changing the position of DBL and RBL. If the RBL were not preset to zero, the data are written to the original data block and conurrently, the RBL is preset to zero, so that it can be used the next time. To track the position of DBL and RBL, additional tag bits are used.

Bishnoi et al. [39] present a technique to reduce write power in STT-RAM by leveraging $1/0$ write asymmetry. As mentioned above, the time taken in a

write is limited by the slower operation and hence, the word line cannot be closed immediately, even when $0 \rightarrow 1$ transition is completed. This leads to significant wastage of power. To address this, they utilize the fact that if the memory cell configuration is going to be 1, then write circuitry can be closed asynchronously as soon as the memory cell is in a stable 1 configuration. Based on this, early write termination is triggered by generating a suitable delayed signal at the time of $0 \rightarrow 1$ write termination.

4.2.5 Improving cache lifetime by wear-leveling

In caches, wear-leveling can be applied at different granularities, viz. color, set, way, memory-cell etc. (see Table 2). On using a runtime wear-leveling technique (WLT), the reconfiguration overhead of a color-level or a set-level WLT is higher than that of a way-level WLT, since the former techniques require change in set-decoding. A memory-cell level WLT does not require change in block-location of a data-item and hence, it incurs negligible overhead, although it cannot address write-variation among different blocks. For any application, on using a fixed capacity and block-size of the cache, increasing the associativity increases the intra-set write-variation and reduces the inter-set/inter-color write-variation. This can be easily understood by considering the two extreme cases, viz. a fully-associative cache and a direct-mapped cache. Hence, for caches with high associativity, intra-set WLTs are more important than inter-set or inter-color WLTs. WLTs may use either in-cache data movement [82, 98] or data invalidation [36, 79, 86, 96]. In-cache data movement may require specialized hardware, while data invalidation increases off-chip accesses, leading to contention and endurance issues in main memory. We now discuss some WLTs and a few other WLTs are discussed in Sections 4.3 and 4.4.

Chen et al. [79] propose a technique for reducing inter-set write-variation to improve the cache lifetime of STT-RAM caches. Their technique aims to remap all set indices after a certain length of time. For this, a register, called remap register is used which is XORed with the set-index bit of the cache address. By periodically changing this register, the set-decoding can be changed. Use of remap register introduces randomization in writes to different cache sets, which helps in uniformly distributing writes to different sets. In a normal cache design, set index bits are not stored as part of the tag. In their scheme, two distinct addresses mapping to different sets can have the same tags and hence, due to remapping, the set index bits need to be included as part of the tag to avoid aliasing.

Mittal [86] proposes a technique to reduce inter-set write variation. His technique uses cache coloring and collects information on the number of writes to each cache color. Periodically, the mapping between physical regions and cache colors is changed so that the regions which see largest number of write operations are mapped to the cache colors which have seen the least number of writes. To avoid aliasing, the affected cache colors are flushed (i.e. dirty blocks are written back and clean blocks are discarded).

4.2.6 Use in GPUs

Maashari et al. [105] investigate use of STT-RAM for designing caches e.g. texture caches and Z caches. They observe that due to high write latency of STT-RAM compared to SRAM, STT-RAM does not always provide performance advantage over SRAM, although due to its low leakage energy consumption, STT-RAM provides better energy efficiency. They also study the impact of 3D stacking of caches on the GPU performance and observe that compared to an iso-cost 2D GPU design, a 3D GPU design offers significant performance advantage due to reduction in cache access latency.

Goswami et al. [92] implement an SRAM-STTRAM hybrid share memory in GPU. In GPUs, shared memory acts as a software-managed cache by allowing the user to manage repetitive data access. Use of STT-RAM enables reduction in leakage power and area. They also use an additional shared memory area which is designed using SRAM that can be configured to work as a cache or RAM. Use of SRAM as a cache to STT-RAM for applications with high amount of write accesses helps in reducing STT-RAM accesses and energy. For other applications, the shared STT-MRAM and SRAM memory reduces write latency. Also, to avoid redundant write operations to STT-RAM, read-before-write based differential memory update (DMU) scheme is used where only the changed cells are written. Thus, their architecture reduces STT-RAM shared memory write access latency using SRAM cache, and also lowers energy consumption by using differential memory update.

4.3 Approaches Using RRAM

Wang et al. [36] present both inter-set and intra-set wear-leveling techniques for RRAM LLCs. The technique for addressing inter-set write-variation aims to distribute the write-traffic evenly to different cache sets by shifting the mapping of cache physical sets to rotate the stored data between sets. Since shifting all cache sets at once would incur a large overhead, their technique swaps only two sets at a time and after multiple swaps, all the cache sets are automatically shifted. The intra-set wear-leveling technique, termed PoLF (probabilistic line-flush) uses data-invalidation approach. After a fixed number of write-hits in the entire cache, PoLF flushes a cache block which sees write-hit, without updating the LRU-age information. Probabilistically, the block being selected for flushing is expected to be hot and thus, the next time, this data-item will be loaded into a cold-block, which leads to wear-leveling. However, due to probabilistic

nature of flushing, PoLF may not always choose an actual hot-block and may lead to large flushes for applications with low write-variation and high write-intensity. Mittal et al. [96] propose using per-block counters for recording the number of writes and flushing a block whose write counter has reached a fixed threshold. By virtue of flushing hot data-items, it leads to better wear-leveling and smaller performance overhead, although it also requires larger number of counters.

Mittal et al. [98] present an in-cache data-movement based technique for mitigating intra-set write variation. Their technique logically divides the cache-sets into multiple modules, for example, in a cache with 4096 sets and 32 modules, each module contains 128 sets. For each module, the number of writes in each way for any of the sets is collectively recorded. Afterwards, their technique makes most frequently written ways in a module unavailable to shift the write-pressure to other ways in the sets of the module. This helps in achieving wear-leveling. Its limitation, however, is that the most frequently written ways in different sets of a module may not have the same index, which limits the effectiveness of wear-leveling.

4.4 Approaches Using PCM

Joo et al. [37] propose several schemes to enable use of PCM for designing on-chip caches. To mitigate the limited write endurance problem of PCM, they propose schemes to minimize the number of writes and distribute them evenly to different memory cells in a cache block. Since read operation in a PCM is much cheaper than the write operation, their write minimization scheme first reads a value before writing, and the new value is written only if it is different from the existing value. The data inverting scheme writes a value in inverted form if it requires less number of bits to be written. They present a data-inverting scheme where the cache block is logically divided into sub-blocks and the decision to write either actual or inverted value to a subblock is taken based on which value leads to smaller number of bit changes. Since this and similar schemes may lead to write-variation in different memory-cells of a block, Joo et al. [37] also propose a memory cell-level WLT. This technique uses a bit-line shifter to spread out the writes over all the memory cells in a cache block to achieve wear-leveling. A bit-line shifter decides the number of bits by which input data is shifted before being written.

4.5 Approaches Using DWM

Venkatesan et al. [91] propose circuit and architecture level techniques for optimizing the DWM-based cache hierarchy. They propose design of two bit cells, namely TAPESTRI-1bit and TAPESTRI-multi. The TAPESTRI-1bit is optimized for performance and unlike conventional DWM cells, does not require any shift operations and hence, can be used in L1 cache. The TAPESTRI-multi bit-cell is optimized for density and requires shift operations before read/write accesses. Using these, they propose a cache hiearchy design where both the tag and data arrays in the L1 cache are designed using TAPESTRI-1bit, while the L2 cache utilizes TAPESTRI-multi for the data array and TAPESTRI-1bit for the tag array. Further, to mitigate the performance overhead caused due to the shift operations in TAPESTRI-multi, they leverage cache pre-shifting approach, where the bits in each cell are predictively shifted such that the bit that is expected to be accessed next is aligned with the read/write port. Using this, the extra latency incurred due to shift operations can be hidden.

Sun et al. [102] propose techniques for architecting DWM caches which aim to minimize the shift cost of DWM. Sun et al. propose two track shifting policies. In the first policy, after the completion of an access, the track stays at its current position and in the second policy, the track returns to its original position. The first policy is more suitable for applications with strong spatial locality, while the second policy facilitates easy data management and is suitable for applications with randomly distributed cache accesses. They also propose a data management policy, where by tracing the data access pattern, the cache blocks with intensive accesses are identified and then placed into the physical locations at the read/write port. This reduces the shifting overhead since in most applications, only a small portion of cache blocks are frequently accessed.

5 Hybrid Approaches

Several researchers propose cache designs which use multiple memory technologies to get the best of them. Although the physical material and read/write properties of different memory technologies are different, the similarity in the peripheral circuits allows similar operation from a logic designer's point of view. Still, the limitations of hybrid designs is that from the perspective of manufacturing, they may incur higher cost of integration, verification, and testing than homogeneous caches. Comparing different hybrid caches, we find that SRAM+PCM caches offer higher area efficiency than SRAM+eDRAM and SRAM+STTRAM caches, however, an important limitation of SRAM+PCM hybrid caches is that the difference in write latency of SRAM and PCM is very large, and hence, a hit in different portions of a hybrid cache may lead to vastly different latencies. Variable hit latency adds complications and unpredictability to scheduling of dependent instructions [4]. Also, the challenges discussed in Section 2 for eDRAM/NVM also apply to the hybrid caches designed using those technologies. We now discuss the key ideas of some of the techniques.

5.1 SRAM+eDRAM Hybrid Caches

Lira et al. [40] propose a NUCA (non-uniform cache architecture) design where different cache banks are designed using SRAM and eDRAM. Their design is based on the observation that a large percentage of cache blocks entering the LLC are not accessed again before they are evicted [11]. Based on this, they propose a placement scheme, where the re-accessed data blocks are stored in fast SRAM banks and the blocks which just arrive to the cache or were demoted from a SRAM bank are stored in large but slow eDRAM banks.

Valero et al. [41] propose a macro-cell design that combines eDRAM and SRAM at cell level. They implement an N-way set-associative cache with these macro-cells consisting of one SRAM cell and N-1 eDRAM cells. At architecture level, they devise a cache access mechanism which attempts to mitigate the destructive-read problem of eDRAM. On a cache access, the tags of all ways and the data of the static cell is read. If there is a hit in the static cell, a delay of accessing only a direct-mapped cache is incurred. If there is a hit in a dynamic cell (as determined by tag matching), the data are accessed with additional delay. To increase the probability of hits in static cell, their technique attempts to always keep the MRU block in static cell using a swap operation.

For a hybrid SRAM-eDRAM last level cache, Valero et al. [43] propose an approach for finding the optimal number of ways to implement using each of the memory technology. Compared to the L1 cache, an L2 cache exhibits much poorer data locality and hence, a hybrid cache design approach where only one of ways is designed using SRAM may not provide optimal performance and energy efficiency. To address this, they suggest using SRAM for designing a larger (e.g. 8 out of 16) number of ways so that a large fraction of cache hits can be served from the SRAM cells.

5.2 SRAM+STTRAM Hybrid Caches

Sharifi et al. [84] propose a cache partitioning approach for providing QoS to different programs in a chip multiprocessor, where the last level shared cache is designed as a hybrid SRAM - STTRAM cache. The SRAM region provides smaller capacity with fast access speed, while opposite is true for the STT-RAM region. They use a control theory centric approach to periodically determine the number of cache ways from both SRAM and STT-RAM portions, which should be allocated to each processor such that its QoS target can be met. To improve performance, they also use a scheme which migrates the frequently accessed blocks from STT-RAM to SRAM.

Li et al. [7] propose an SRAM - STTRAM hybrid cache design to increase cache capacity and mitigate the limited write endurance problem of STT-RAM. They use compiler and operating system support to migrate or swap the write intensive data from STT-RAM to SRAM. The compiler identifies the write reuse patterns of heap data objects such as linked structures and arrays and inserts instructions to guide the hardware to perform the swap/migration dynamically.

Chen et al. [12] propose a dynamic cache reconfiguration technique for saving energy in a hybrid SRAM - STTRAM cache. A few cache ways are designed using SRAM while most other ways are designed using STTRAM. Their technique tracks the frequency of accesses to different cache ways and turns-off those ways for which the access frequency in an interval falls below a certain threshold.

Li et al. [103] propose a way-based hybrid SRAM - STTRAM cache design. The SRAM is used to accommodate write-intensive cache blocks. Since applications exhibit significant inter-set variation in their write intensity, using the SRAM ways exclusively for its own cache set leads to inefficient use of SRAM capacity. To address this, they propose organizing the SRAM blocks in the hybrid cache as a semi-independent set-associative cache. Using this, several hybrid cache sets can efficiently share and cooperatively utilize their SRAM blocks.

Quan et al. [55] propose an SRAM - STTRAM hybrid cache design that aims to mitigate the long write latency and power problem of STT-RAM. Their technique uses a prediction table that records the access information about each cache line, which is used to predict which lines in SRAM have become dead and which lines are less or more frequently written. Using this information, their technique replaces the dead cache lines in SRAM to improve its utilization ratio. Also, the frequently written lines are written into SRAM and less-frequently written lines are replaced from SRAM as soon as possible. The replaced dead lines are inserted in the STT-RAM. Thus, their technique saves energy by minimizing the number of writes to the STT-RAM.

Chen et al. [69] propose an SRAM - STTRAM hybrid cache design which aims to minimize the number of writes to the STT-RAM. Their cache management technique uses the compiler to provide static hints to guide initial data placement such that the write-intensive data are placed into SRAM and non-write-intensive data are placed into STTRAM. Since compiler hints may be misleading due to lack of runtime information and input variation, they also use hardware-support to correct the hints provided by the compiler at runtime. Specifically, if a write-intensive block is placed in the STT-RAM, it is swapped with an SRAM block which shows smaller amount of write intensity.

Ahn et al. [89] propose a write-intensity predictor for reducing the number of writes to STT-RAM in an SRAM - STTRAM hybrid cache. They note that the data accessed by the same load/store instruction

shares similar characteristics, thus, the number of writes per cache block is correlated with the trigger instruction that loaded the block into the cache. Based on this, their technique keeps track of the instructions that tend to load write-intensive blocks and utilizes this information to predict the write-intensity of the blocks that will be accessed in the future. Using this, the write-intensive blocks are loaded into SRAM region, while other blocks are loaded into SRAM region, which leads to saving the write energy.

Li et al. [58] propose a compiler based approach for reducing the migrations in a hybrid SRAM - STTRAM cache. The cache management policies for hybrid caches aim to migrate the write-intensive blocks from STT-RAM to SRAM, however, this requires additional read and write operations and leads to significant overheads. Li et al. observe that migrations are sensitive to data access patterns, and correlate closely with the transition events where a read operation is followed by a write operation, or vice versa. They also observe that in embedded systems, most of such transition events come from stack area and global area, rather than heap area. To reduce the migration overhead, they propose a migration-aware compilation approach which places the data with consecutively same access operations into the same memory block. This reduces the transition events which also reduces the number of migrations and improves the energy efficiency.

Zhao et al. [14] propose a hybrid cache hierarchy where each level is designed with a memory technology such that the bandwidth provided by the overall hierarchy is optimized. They study the write endurance, latency, energy and bandwidth of different technologies. Since the access power of a cache is approximately proportional to $bandwidth \times \sqrt{capacity}$ [113], under the energy constraint, increasing the capacity leads to reduction in the bandwidth. Thus, each memory technology provides the suitable bandwidth only in a range of capacity. Based on this, they select SRAM, STT-RAM and eDRAM for designing L2, L3 and L4 caches, respectively.

5.3 SRAM+PCM Hybrid Caches

Guo et al. [100] propose a way-level hybrid SRAM-PCM cache design which aims to leverage the high capacity of PCM while keeping its write-traffic low to extend its lifetime. Their technique detects both dead and write-intensive cache lines. The dead cache lines are evicted as soon as possible and write-intensive cache lines are mapped to SRAM or written back to memory. This reduces the number of writes to PCM which also leads to saving of energy.

Wu et al. [10] propose a hybrid cache design where L2 cache consists of a small fast region designed with SRAM and another large slow region designed with eDRAM or STT-RAM or PCM. To manage this cache, they propose suitable cache line replacement and data migration policies. Their technique records the access frequency of lines in slow region and when it exceeds a threshold, the line is swapped with another line in the fast region. To save energy, they also propose keeping the slow region in state-preserving low-leakage (drowsy) mode.

Wu et al. [10] also propose inter-level hybrid cache hierarchies, where the choice of memory technology for designing each cache level is done based on their properties. L1 and L2 are designed using SRAM due to its performance advantage. In one hybrid cache design, L3 is designed using eDRAM, STT-RAM or PCM. In another hybrid cache design, L3 is designed using eDRAM or STT-RAM and L4 is designed using PCM due to its slow speed and high density. They have shown that their hybrid cache design significantly improves over a pure-SRAM cache hierarchy.

5.4 SRAM+DWM Hybrid Caches

To leverage the best of both SRAM and DWM, Venkatesan et al. [18] propose a cache design with DWM based data array and an SRAM based tag array. They note that mapping all the bits in a cache block to the same DWM tape requires N serial read/write operations for accessing N bits in the DWM tape, which leads to very high latency. To reduce the impact of shift latencies, they propose a bit-interleaved cache organization in which each cache block is spread across several DWM macro-cells. As an example, N 64-bit blocks are stored in 64 DWM tapes each containing N bits of data.

6 Conclusion

Emerging memory technologies such as eDRAM, STT-RAM, RRAM, PCM and DWM have several desirable properties such as low-leakage power and high density and hence, they provide new opportunities in system design. However, they also have various shortcomings such as limited write endurance, high write latency etc. which need to be overcome. In this paper, we presented a survey of architectural techniques for addressing the issues in caches designed with emerging memory technologies. We also presented a classification of the techniques based on important parameters. We believe that these emerging technologies and techniques will become increasingly effective and integrated in the mainstream processors.

References

[1] P. Kogge *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," DARPA Information Processing Techniques Office, Tech. Rep., 2008.
[2] J. Dongarra *et al.*, "The international exascale software project roadmap," *IJHPCA*, vol. 25, no. 1, pp. 3–60, 2011.
[3] Intel, http://ark.intel.com/products/53575/.
[4] S. Mittal, "A survey of architectural techniques for improving cache power efficiency," *Elsevier Sustainable Computing: Informatics and Systems*, vol. 4, no. 1, pp. 33–43, 2014.

[5] B. M. Rogers *et al.*, "Scaling the bandwidth wall: challenges in and avenues for CMP scaling," *ISCA*, pp. 371–382, 2009.
[6] M. K. Qureshi, S. Gurumurthi, and B. Rajendran, "Phase change memory: From devices to systems," *Synthesis Lectures on Computer Architecture*, vol. 6, no. 4, pp. 1–134, 2011.
[7] Y. Li *et al.*, "A software approach for combating asymmetries of non-volatile memories," in *ISLPED*, 2012, pp. 191–196.
[8] S.-M. Syu *et al.*, "High-endurance hybrid cache design in CMP architecture with cache partitioning and access-aware policy," in *GLSVLSI*, 2013.
[9] S. Mittal, "Energy Saving Techniques for Phase Change Memory (PCM)," Iowa State University, USA, Tech. Rep., 2013.
[10] X. Wu *et al.*, "Hybrid cache architecture with disparate memory technologies," *ISCA*, pp. 34–45, 2009.
[11] M.-T. Chang *et al.*, "Technology Comparison for Large Last-Level Caches (L^3Cs): Low-Leakage SRAM, Low Write-Energy STT-RAM, and Refresh-Optimized eDRAM," *HPCA*, pp. 143–154, 2013.
[12] Y.-T. Chen *et al.*, "Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design," in *DATE*, 2012, pp. 45–50.
[13] C. W. Smullen *et al.*, "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," in *HPCA*, 2011.
[14] J. Zhao *et al.*, "Bandwidth-aware reconfigurable cache design with hybrid memory technologies," in *ICCAD*, 2010.
[15] X. Dong *et al.*, "Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement," in *DAC*, 2008, pp. 554–559.
[16] S. Mittal *et al.*, "CASHIER: A Cache Energy Saving Technique for QoS Systems," *IEEE VLSID*, pp. 43–48, 2013.
[17] M. H. Kryder *et al.*, "After hard drives–what comes next?" *IEEE Trans. on Magnetics*, vol. 45, no. 10, pp. 3406–3413, 2009.
[18] R. Venkatesan *et al.*, "TapeCache: a high density, energy efficient cache based on domain wall memory," in *ISLPED*, 2012, pp. 185–190.
[19] M. Rasquinha *et al.*, "An energy efficient cache design using spin torque transfer (STT) RAM," in *ISLPED*, 2010.
[20] http://www.mram-info.com/companies.
[21] http://www.memorystrategies.com/report/embeddeddram.html.
[22] http://www.economist.com/node/21560981.
[23] R. Kalla *et al.*, "Power7: IBM's next-generation server processor," *Micro, IEEE*, vol. 30, no. 2, pp. 7–15, 2010.
[24] S. Iyer *et al.*, "Embedded DRAM: Technology platform for the Blue Gene/L chip," *IBM JRD*, pp. 333–350, 2005.
[25] N. Kurd *et al.*, "Haswell: A family of IA 22nm processors," in *IEEE ISSCC*, 2014, pp. 112–113.
[26] A. Agrawal *et al.*, "Refrint: Intelligent refresh to minimize power in on-chip multiprocessor cache hierarchies," *HPCA*, pp. 400 – 411, 2013.
[27] CACTI 5.3, http://quid.hpl.hp.com:9081/cacti/.
[28] J. Barth *et al.*, "A 500 MHz random cycle, 1.5 ns latency, SOI embedded DRAM macro featuring a three-transistor micro sense amplifier," *IEEE JSSC*, vol. 43, no. 1, pp. 86–95, 2008.
[29] C. Wilkerson *et al.*, "Reducing cache power with low-cost, multi-bit error-correcting codes," *ISCA*, pp. 83–93, 2010.
[30] M. Alizadeh *et al.*, "Versatile refresh: low complexity refresh scheduling for high-throughput multi-banked eDRAM," in *ACM SIGMETRICS PER*, vol. 40, no. 1, 2012, pp. 247–258.
[31] Y. Huai, "Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects," *AAPPS Bulletin*, vol. 18, no. 6, pp. 33–40, 2008.
[32] A. Jog *et al.*, "Cache revive: architecting volatile STT-RAM caches for enhanced performance in CMPs," in *DAC*, 2012, pp. 243–252.
[33] H. Li and Y. Chen, "An overview of non-volatile memory technology and the implication for tools and architectures," in *DATE*, 2009, pp. 731–736.
[34] Y.-B. Kim *et al.*, "Bi-layered RRAM with unlimited endurance and extremely uniform switching," in *VLSIT*, 2011, pp. 52–53.
[35] X. Dong *et al.*, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE TCAD*, vol. 31, no. 7, pp. 994–1007, 2012.
[36] J. Wang *et al.*, "i^2WAP: Improving non-volatile cache lifetime by reducing inter-and intra-set write variations," in *HPCA*, 2013, pp. 234–245.
[37] Y. Joo *et al.*, "Energy-and endurance-aware design of phase change memory caches," in *DATE*, 2010, pp. 136–141.
[38] R. Bishnoi *et al.*, "Avoiding Unnecessary Write Operations in STT-MRAM for Low Power Implementation," *ISQED*, 2014.
[39] R. Bishnoi *et al.*, "Asynchronous asymmetrical write termination (AAWT) for a low power STT-MRAM," in *DATE*, 2014.
[40] J. Lira *et al.*, "Implementing a hybrid SRAM/eDRAM NUCA architecture," in *IEEE HiPC*, 2011, pp. 1–10.
[41] A. Valero *et al.*, "An hybrid eDRAM/SRAM macrocell to implement first-level data caches," in *MICRO*, 2009.
[42] A. Valero *et al.*, "Exploiting reuse information to reduce refresh energy in on-chip eDRAM caches," in *ICS*, 2013, pp. 491–492.
[43] A. Valero *et al.*, "Analyzing the optimal ratio of SRAM banks in hybrid caches," in *IEEE ICCD*, 2012, pp. 297–302.
[44] W. R. Reohr, "Memories: Exploiting them and developing them," in *IEEE SOCC*, 2006, pp. 303–310.
[45] S. Mittal, "A Cache Reconfiguration Approach for Saving Leakage and Refresh Energy in Embedded DRAM Caches," Iowa State University, Iowa, USA, Tech. Rep., 2013.
[46] V. Lorente *et al.*, "Combining RAM technologies for hard-error recovery in L1 data caches working at very-low power modes," in *DATE*, 2013, pp. 83–88.
[47] S. Mittal *et al.*, "Improving energy efficiency of Embedded DRAM Caches for High-end Computing Systems," in *ACM HPDC*, 2014.
[48] A. Agrawal *et al.*, "Mosaic: Exploiting the Spatial Locality of Process Variation to Reduce Refresh Energy in On-Chip eDRAM Modules," in *HPCA*, 2014.
[49] L. Jiang *et al.*, "Constructing large and fast multi-level cell STT-MRAM based cache for embedded processors," in *DAC*, 2012, pp. 907–912.
[50] J. Ahn and K. Choi, "Lower-bits cache for low power STT-RAM caches," in *ISCAS*, 2012, pp. 480–483.
[51] J. Wang, X. Dong, and Y. Xie, "OAP: an obstruction-aware cache management policy for STT-RAM last-level caches," in *DATE*, 2013, pp. 847–852.
[52] X. Bi, Z. Sun, H. Li, and W. Wu, "Probabilistic design methodology to improve run-time stability and performance of STT-RAM caches," in *ICCAD*, 2012, pp. 88–94.
[53] J. Jung *et al.*, "Energy-efficient Spin-Transfer Torque RAM cache exploiting additional all-zero-data flags," in *ISQED*, 2013, pp. 216–222.
[54] Q. Li *et al.*, "Compiler-assisted refresh minimization for volatile STT-RAM cache," in *ASP-DAC*, 2013, pp. 273–278.
[55] B. Quan *et al.*, "Prediction Table based Management Policy for STT-RAM and SRAM Hybrid Cache," *ICCCT*, pp. 1092 – 1097, 2012.
[56] S. Yazdanshenas *et al.*, "Coding Last Level STT-RAM Cache For High Endurance And Low Power," *IEEE CAL*, 2013.
[57] J. Li, C. J. Xue, and Y. Xu, "STT-RAM based energy-efficiency hybrid cache for CMPs," in *VLSI-SoC*, 2011, pp. 31–36.
[58] Q. Li *et al.*, "Mac: Migration-aware compilation for STT-RAM based hybrid cache in embedded systems," in *ISLPED*, 2012, pp. 351–356.
[59] Q. Li *et al.*, "Compiler-assisted preferred caching for embedded systems with STT-RAM based hybrid cache," *ACM SIGPLAN Notices*, vol. 47, no. 5, pp. 109–118, 2012.
[60] Z. Sun *et al.*, "Multi retention level STT-RAM cache designs with a dynamic refresh scheme," in *MICRO*, 2011, pp. 329–338.
[61] G. Sun *et al.*, "A novel architecture of the 3D stacked MRAM L2 cache for CMPs," in *HPCA*, 2009, pp. 239–249.
[62] H. Sun *et al.*, "Design techniques to improve the device write margin for MRAM-based cache memory," in *GLSVLSI*, 2011, pp. 97–102.
[63] X. Bi, H. Li, and J.-J. Kim, "Analysis and Optimization of Thermal Effect on STT-RAM Based 3-D Stacked Cache Design," in *IEEE ISVLSI*, 2012, pp. 374–379.
[64] Y. Joo and S. Park, "A Hybrid PRAM and STT-RAM Cache Architecture for Extending the Lifetime of PRAM Caches," *IEEE CAL*, 2012.
[65] B.-M. Lee and G.-H. Park, "Performance and energy-efficiency analysis of hybrid cache memory based on SRAM-MRAM," in *ISOCC*, 2012, pp. 247–250.
[66] M. Mao *et al.*, "Coordinating prefetching and STT-RAM based last-level cache management for multicore systems," in *GLSVLSI*, 2013, pp. 55–60.
[67] Y. Li and A. K. Jones, "Cross-layer techniques for optimizing

systems utilizing memories with asymmetric access characteristics," in *IEEE ISVLSI*, 2012, pp. 404–409.
[68] H. Sun *et al.*, "Using magnetic RAM to build low-power and soft error-resilient L1 cache," *IEEE TVLSI*, vol. 20, no. 1, pp. 19–28, 2012.
[69] Y.-T. Chen *et al.*, "Static and dynamic co-optimizations for blocks mapping in hybrid caches," in *ISLPED*, 2012.
[70] G. Sun *et al.*, "Exploring the vulnerability of CMPs to soft errors with 3D stacked non-volatile memory," in *ICCD*, 2011, pp. 366–372.
[71] H. Naeimi *et al.*, "STTRAM scaling and retention failure," *Intel Technology Journal*, vol. 17, no. 1, p. 54, 2013.
[72] J. Li *et al.*, "Cache coherence enabled adaptive refresh for volatile STT-RAM," in *DATE*, 2013, pp. 1247–1250.
[73] J. Ahn, S. Yoo, and K. Choi, "Selectively protecting error-correcting code for area-efficient and reliable STT-RAM caches," in *ASP-DAC*, 2013, pp. 285–290.
[74] Z. Sun *et al.*, "Process variation aware data management for STT-RAM cache design," in *ISLPED*, 2012, pp. 179–184.
[75] Y. Zhou *et al.*, "Asymmetric-access aware optimization for STT-RAM caches with process variations," in *GLSVLSI*. ACM, 2013, pp. 143–148.
[76] S. Lee *et al.*, "Hybrid cache architecture replacing SRAM cache with future memory technology," in *ISCAS*, 2012, pp. 2481–2484.
[77] V. Saripalli *et al.*, "Exploiting heterogeneity for energy efficiency in chip multiprocessors," *IEEE JETCAS*, vol. 1, no. 2, pp. 109–119, 2011.
[78] H. Li *et al.*, "Performance, power, and reliability tradeoffs of STT-RAM cell subject to architecture-level requirement," *IEEE Trans. Magn.*, vol. 47, no. 10, pp. 2356–2359, 2011.
[79] Y. Chen *et al.*, "On-chip caches built on multilevel spin-transfer torque RAM cells and its optimizations," *J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 2, pp. 16:1–16:22, 2013.
[80] P. Zhou *et al.*, "Energy reduction for STT-RAM using early write termination," in *ICCAD*, 2009, pp. 264–268.
[81] S. P. Park *et al.*, "Future cache design using STT MRAMs for improved energy efficiency: devices, circuits and architecture," in *DAC*, 2012, pp. 492–497.
[82] A. Jadidi *et al.*, "High-endurance and performance-efficient design of hybrid cache architectures through adaptive line replacement," in *ISLPED*, 2011, pp. 79–84.
[83] L. V. Cargnini *et al.*, "Embedded memory hierarchy exploration based on magnetic RAM," in *FTFC*, 2013, pp. 1–4.
[84] A. Sharifi and M. Kandemir, "Automatic Feedback Control of Shared Hybrid Caches in 3D Chip Multiprocessors," in *Euromicro PDP*, 2011, pp. 393–400.
[85] K. Swaminathan *et al.*, "Design space exploration of workload-specific last-level caches," in *ISLPED*, 2012, pp. 243–248.
[86] S. Mittal, "Using cache-coloring to mitigate inter-set write variation in non-volatile caches," Iowa State University, Tech. Rep., 2013.
[87] J. Ahn and K. Choi, "LASIC: Loop-Aware Sleepy Instruction Caches Based on STT-RAM Technology," *IEEE TVLSI*, 2013.
[88] K.-W. Kwon *et al.*, "AWARE (Asymmetric Write Architecture with REdundant blocks): A High Write Speed STT-MRAM Cache Architecture," *IEEE TVLSI*, 2013.
[89] J. Ahn *et al.*, "Write intensity prediction for energy-efficient non-volatile caches," in *ISLPED*, 2013, pp. 223–228.
[90] Z. Sun *et al.*, "A dual-mode architecture for fast-switching STT-RAM," in *ISLPED*, 2012, pp. 45–50.
[91] R. Venkatesan *et al.*, "DWM-TAPESTRI-an energy efficient all-spin cache using domain wall shift based writes," in *DATE*, 2013, pp. 1825–1830.
[92] N. Goswami *et al.*, "Power-performance co-optimization of throughput core architecture using resistive memory," in *HPCA*, 2013, pp. 342–353.
[93] Y. Li *et al.*, "C1C: A configurable, compiler-guided STT-RAM L1 cache," *ACM TACO*, vol. 10, no. 4, pp. 52:1–52:22, 2013.
[94] J. Zhao and Y. Xie, "Optimizing bandwidth and power of graphics memory with hybrid memory technologies and adaptive data migration," in *ICCAD*, 2012, pp. 81–87.
[95] N. Strikos *et al.*, "Low-current probabilistic writes for power-efficient STT-RAM caches," in *ICCD*, 2013, pp. 511–514.
[96] S. Mittal *et al.*, "LastingNVCache: A Technique for Improving the Lifetime of Non-volatile Caches," in *ISVLSI*, 2014.
[97] X. Dong *et al.*, "A circuit-architecture co-optimization framework for evaluating emerging memory hierarchies," in *ISPASS*, 2013, pp. 140–141.
[98] S. Mittal *et al.*, "WriteSmoothing: Improving Lifetime of Non-volatile Caches Using Intra-set Wear-leveling," in *ACM GLSVLSI*, 2014.
[99] P. Mangalagiri *et al.*, "A low-power phase change memory based hybrid cache architecture," in *GLSVLSI*, 2008, pp. 395–398.
[100] S. Guo *et al.*, "Wear-resistant hybrid cache architecture with phase change memory," in *IEEE NAS*, 2012, pp. 268–272.
[101] M. Sharad *et al.*, "Multi-level magnetic RAM using domain wall shift for energy-efficient, high-density caches," in *ISLPED*, 2013, pp. 64–69.
[102] Z. Sun *et al.*, "Cross-layer racetrack memory design for ultra high density and low power consumption," in *DAC*, 2013.
[103] J. Li *et al.*, "Exploiting set-level write non-uniformity for energy-efficient NVM-based hybrid cache," in *IEEE ESTIMedia*, 2011, pp. 19–28.
[104] H. Noguchi *et al.*, "D-MRAM cache: enhancing energy efficiency with 3T-1MTJ DRAM/MRAM hybrid memory," in *DATE*, 2013, pp. 1813–1818.
[105] A. Al Maashri *et al.*, "3D GPU architecture using cache stacking: Performance, cost, power and thermal analysis," in *ICCD*, 2009, pp. 254–259.
[106] P. Satyamoorthy, "STT-RAM for Shared Memory in GPUs," Ph.D. dissertation, University of Virginia, 2011.
[107] X. Guo *et al.*, "Resistive computation: avoiding the power wall with low-leakage, STT-MRAM based computing," *ISCA*, pp. 371–382, 2010.
[108] X. Bi *et al.*, "Unleashing the potential of MLC STT-RAM caches," in *ICCAD*. IEEE Press, 2013, pp. 429–436.
[109] Y. Xie, "Future memory and interconnect technologies," in *DATE*, 2013, pp. 964–969.
[110] S. Kaxiras *et al.*, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *ISCA*, 2001, pp. 240–251.
[111] S. Mittal *et al.*, "FlexiWay: A Cache Energy Saving Technique Using Fine-grained Cache Reconfiguration," in *ICCD*, 2013.
[112] S. Mittal *et al.*, "MASTER: A Multicore Cache Energy Saving Technique using Dynamic Cache Reconfiguration," *IEEE TVLSI*, 2013.
[113] G. Sun *et al.*, "Moguls: a model to explore the memory hierarchy for bandwidth improvements," in *ISCA*, 2011, pp. 377–388.

Sparsh Mittal received the B.Tech. degree in electronics and communications engineering from IIT, Roorkee, India and the Ph.D. degree in computer engineering from Iowa State University, USA. He is currently working as a Post-Doctoral Research Associate at ORNL. His research interests include non-volatile memory, memory system power efficiency, cache and GPU architectures.

Jeffrey S. Vetter , Ph.D., holds a joint appointment between ORNL and Georgia Institute of Technology (GT). At ORNL, he is a Distinguished R&D Staff Member, and the founding group leader of the Future Technologies Group. At GT, he is a Joint Professor in the Computational Science and Engineering School, the Project Director for the NSF Track 2D Experimental Computing Facility for large scale heterogeneous computing using graphics processors, and the Director of the NVIDIA CUDA Center of Excellence. His research interests include massively multithreaded processors, non-volatile memory and heterogeneous multicore processors.

Dong Li received the Ph.D. degree in Computer Science from Virginia Tech, USA. He is currently a research staff member at ORNL and an adjunct assistant professor in the Department of Electrical Engineering and Computer Science at the University of Tennessee at Knoxville. His research interests include resilience, power-aware computing, performance modeling and optimization, and memory systems.