Mellow Writes: Extending Lifetime in Resistive Memories through Selective Slow Write Backs

Lunkai Zhang†, Brian Neely◇, Diana Franklin†, Dmitri Strukov‡, Yuan Xie‡, Frederic T. Chong†
†Department of Computer Science, University of Chicago
◇Department of Computer Science, UC Santa Barbara
‡Electrical and Computer Engineering, UC Santa Barbara
{lunkai, dmfranklin}@uchicago.edu, bkneely@gmail.com,
{strukov, yuanxie}@ece.ucsb.edu, chong@cs.uchicago.edu

***Abstract*—Emerging resistive memory technologies, such as PCRAM and ReRAM, have been proposed as promising replacements for DRAM-based main memory, due to their better scalability, low standby power, and non-volatility. However, limited write endurance is a major drawback for such resistive memory technologies. Wear leveling (balancing the distribution of writes) and wear limiting (reducing the number of writes) have been proposed to mitigate this disadvantage, but both techniques only manage a fixed budget of writes to a memory system rather than *increase* the number available.**

In this paper, we propose a new type of wear limiting technique, *Mellow Writes*, which reduces the wearout of individual writes rather than reducing the number of writes. *Mellow Writes* is based on the fact that slow writes performed with lower dissipated power can lead to longer endurance (and therefore longer lifetimes). For non-volatile memories, an N^1 to N^3 times endurance can be achieved if the write operation is slowed down by N times.

We present three microarchitectural mechanisms (*Bank-Aware Mellow Writes*, *Eager Mellow Writes*, and *Wear Quota*) that selectively perform slow writes to increase memory lifetime while minimizing performance impact. Assuming a factor N^2 advantage in cell endurance for a factor N slower write, our best Mellow Writes mechanism can achieve 2.58× lifetime and 1.06× performance of the baseline system. In addition, its performance is almost the same as a system aggressively optimized for performance (at the expense of endurance). Finally, Wear Quota guarantees a minimal lifetime (e.g., 8 years) by forcing more slow writes in presence of heavy workloads. We also perform sensitivity analysis on the endurance advantage factor for slow writes, from N^1 to N^3, and find that our technique is still useful for factors as low as N^1.

***Keywords*-non-volatile memory, endurance, write latency**

I. Introduction

DRAM technology scaling is fundamentally limited by its use of capacitance to store values, requiring energy wasting refreshes and losing values when power is removed. Emerging resistive memory technologies, such as resistive random access memory (ReRAM) and phase change memory have shown promise as DRAM replacements. Such emerging memory technologies have the advantage of high density, high scalability, non-volatility, and low standby power. They fall short, however, in one category: *write endurance*. Fatigued cells may fail to change state. This often results in data errors, making write endurance a serious challenge for architecting resistive memory systems.

There are two common methods to combat the endurance limit of resistive memories:

- **Wear Leveling.** Most applications exhibit *non-uniform* write patterns, so the resistive memory cells in hotspot memory blocks will have a much shorter lifetime than others. Wear leveling evens out write patterns by remapping heavily written lines to less frequently written lines. The limit of wear leveling is that its lifetime improvement has an upper bound: the average lifetime of memory cells.
- **Wear Limiting.** Wear limiting attempts to reduce–rather than distribute–the amount of wear on the memory system. Some existing techniques are DRAM buffering [1] and Flip-N-Write [2]. All of these function by reducing the number of writes occurring to the resistive devices.

In this paper, we introduce a new technique to implement wear limiting on resistive memories. Instead of reducing the *number* of writes, we reduce the *impact* of some writes on the endurance by performing a slower write. In this paper, we will use an analytic model from [3] and ReRAM parameters, but our techniques are applicable to any systems with variable wear that is correlated with the speed of writes. In fact, our sensitivity analysis will show benefits for write speed to endurance relationships that vary from linear to cubic. The key, however, is to use slow writes strategically to avoid perfromance degradation.

We evaluate two *Mellow Writes* schemes: *Bank-Aware Mellow Writes* and *Eager Mellow Writes*. *Bank-Aware Mellow Writes* inspects the current set of write requests, sending slow writes to banks that have only one current write request. *Eager Mellow Writes* goes one step further, identifying useless dirty lines in the last level cache (LLC) to write back to banks with no requests. Experiments show that our proposed techniques preserve performance and significantly enhance lifetime—achieving almost the same performance as a system with the most aggressive speed-up techniques

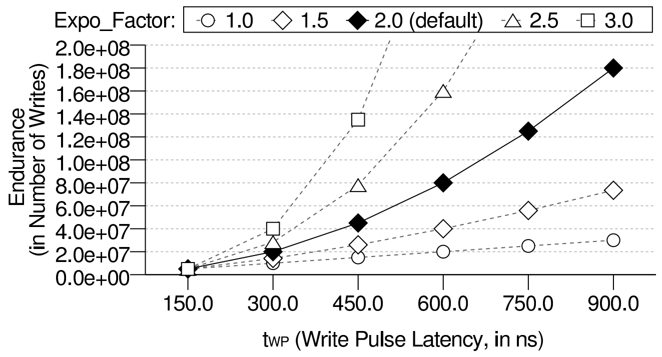

Figure 1. **Trade-off between Write Latency and Endurance in our simulated resistive memory system (see Section V).** Without loss of generality, we use ReRAM technology with the baseline write latency of 150ns and the baseline endurance of $5*10^6$. For slow writes, we model the endurance based on Equation 2 (derived from paper [3]), and use five different *Expo_Factor*: 1.0, 1.5, 2.0, 2.5, 3.0. By default, we chose to model a quadratic write latency / lifetime tradeoff ($Expo\_Factor = 2.0$).

while achieving 2.58× lifetime of a system with a default configuration without aggressive performance optimizations. In addition, we propose a *Wear Quota* to guarantee the mininal lifetime (e.g., 8 years in our experiment) of the ReRAM memory system. Finally, we find that while our scheme is influenced by the exponential relationship between latency and endurance we derive from [3], our scheme is still advantageous with a pessimistic linear model.

The rest of the paper is organized as follows. We first introduce the wear/latency trade-off in resistive memory in Section II. Our motivation section (Section III) shows the system-level performance impact of slow writes and the abundance of memory idle time. Section IV describes our two *Mellow Writes* schemes and the *Wear Quota* scheme. We then present our methodology and results in Sections V and VI. We have a more detailed related work section in Section VII, followed by our conclusions in Section VIII.

II. Write Latency/Endurance Trade-off

The relationship between write latency and endurance is intuitive and has been observed before [4], [5] for different kinds of non-volatile memories. In order to provide low latency in writes, it is typical to apply high power dissipation [6], [7]. Higher power dissipation accelerates failure mechanisms [8], [9], [5].

In this paper, we will use a recent analytic model [3] which makes two observations. First, switching speed is exponentially dependent on electric field and temperature for many types of nonvolatile memories including flash [10], phase change [11], [12], magnetoresistive and ferroelectric [13], and ReRAMs [14], [9] memories. Second, a high electric field combined with a high temperature exponentially increases the probability of creating new defects and/or filling existing deep traps in the dielectric [8] which is a primary source for limited endurance in majority of non-volatile memories (e.g. those relying on electron-tunneling phenomena).

Combining these two observations, the analytic model [3] derives a polynomial relationship between endurance and write latency. Specifically, a cell's endurance (in terms of total amount of writes) is given by:

$$Endurance \approx (\frac{t_{WP}}{t_0})^{\frac{U_F}{U_S}-1} \tag{1}$$

where t_{WP} is write latency, t_0 is a device related constant, U_F is the activation energy for failure mechanism, and U_S is the activation energy of switching mechanism.

For non-volatile memories, practical values of U_S should be above 1eV [15]. Assuming practical values for U_F [8], [16], $\frac{U_F}{U_S}$ ranges from 2 to 4, so that, e.g., latency decrease is linearly, quadratically, and cubically proportional to endurance decrease for $\frac{U_F}{U_S}=2$, $\frac{U_F}{U_S}=3$ and $\frac{U_F}{U_S}=4$, respectively. Thus our endurance model becomes:

$$Endurance \approx (\frac{tWP}{t_0})^{Expo\_Factor} \tag{2}$$

where $1 \leqslant Expo\_Factor \leqslant 3$.

As is shown in Figure 1, without loss of generality, we model a resistive memory with the baseline write latency of 150ns and the baseline endurance of $5*10^6$. Our baseline experiments model a quadratic write latency / lifetime trade-off ($Expo\_Factor = 2.0$). This corresponds to $U_F \gtrsim 3eV$ which is representative of energy for creating a vacancy in many relevant metal oxide devices [17]. Our sensitivity experiments, however, model five different *Expo_Factor*: 1.0, 1.5, 2.0, 2.5, 3.0.

III. Motivation

The motivation for our solution is based on two observations. First, using a single write latency cannot fulfill the performance and lifetime requirements for varying workloads. Second, for a system with a typical write latency, the memory banks are idle for much of the time.

In order to demonstrate the impact that different write latencies have on performance and lifetime, we run our baseline system (see Section V for details) with four different write latencies: normal write (1.0× latency) and slow writes (1.5×, 2.0× and 3.0× normal write latency). In addition, the use of write cancellations [18], [19] (in the presence of a read to the same bank) may also influence both performance and lifetime, so we perform simulations with and without write cancellation.

Figure 2 shows the results. We make the following important observations:

- Short latency (e.g., 1.0-1.5× normal writes) leads to unreasonably short lifetimes for some benchmarks (e.g., `lbm`, `leslie3d`, etc.).
- Although slow writes provide longer lifetime, they can considerably degrade the overall system performance (`stream`: 63.8% degradation at 3.0×, 30.1% at 1.5×). Therefore, slow writes must be used judiciously.

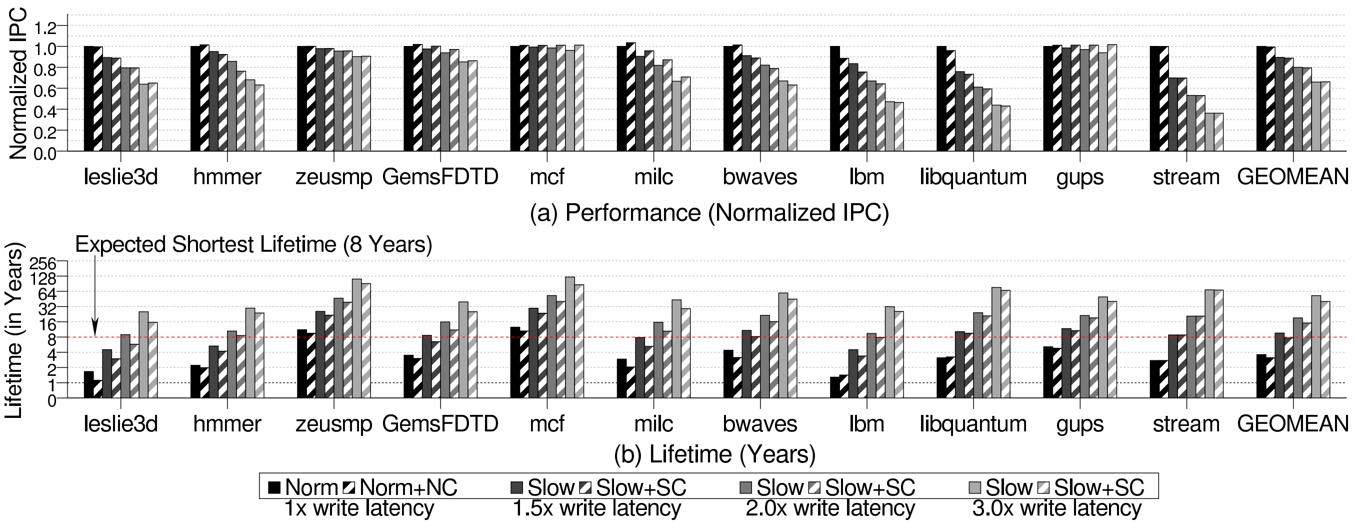

Figure 2. **Normalized IPC and Lifetime (in years) of systems with normal writes and 1.5×–3.0× slow writes.**

- Write cancellation is no silver bullet for slow writes. It helps by allowing reads to complete more quickly (`milc, mcf`), but can degrade performance by putting pressure on write queues, leading to more of the expensive write drain operations(`hmmer, bwaves`). Write cancellation also comes at a penalty to memory lifetime due to the multiple write attempts. Therefore, write cancellation is only part of the solution to reduce the performance penalty of slow writes.

In summary, to get higher performance on the premise of guaranteeing minimal lifetime (in our case, 8 years), different applications favor different write latencies and write policies. For example, `leslie3d` favors 2× latency writes without write cancellation; `gups` is suitable for 1.5× latency writes with write cancallation; and `zeusmp` likes 1× latency writes without write cancellation. As a result, it is hard, if not impossible, to find a fixed write latency and a fixed write policy to fit all the applications.

Therefore, it is necessary to use adaptive write schemes in which a system can adaptively use fast and slow writes. The former ones improve the performance, and the latter ones extend the lifetime. Figure 3 shows the opportunity—in a system with fast writes (i.e., with 1.0× latency), the utilization of memory banks is not particularly high. Therefore, there are ample opportunities to selectively write dirty data using slower writes (e.g., with 3.0× latency) when their corresponding banks are not busy.

Our goal is to use slow writes at times when it is least likely to lead to performance degradation. Intuitively, we will be using the LLC as a large buffer from which we can find proper cachelines to be eagerly and slowly written back during periods of memory idle time. Predicting which cachelines are proper to do so, and when to perform the eager slow writes, is the challenge we face.

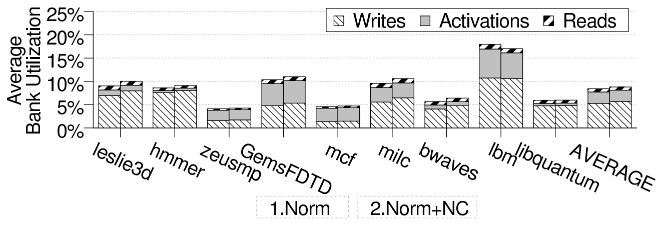

Figure 3. **Average bank utilization of systems with normal writes. The utilization of a bank refers to the percentage of the time when corresponding bank is busy.**

IV. Mellow Writes

In this section, we discuss how to adaptively use fast and slow writes. For hardware simplicity consideration, in this paper we just adaptively use two different kinds of writes: normal writes with 1× latency, and slow writes with 3× latency.

We introduce our two *Mellow Writes* schemes that selectively perform slow writes when the memory system is relatively less busy. Both schemes depend on idle memory cycles to perform such writes. In order to protect against memory-intensive workloads that do not have enough idle times, we also introduce a *Wear Quota* scheme to provide guaranteed lifetimes (at the cost of performance).

A. Bank-Aware Mellow Writes

Bank-Aware Mellow Writes scheme exploits the fact that a program could have many writes and still have idle time in a particular bank. Therefore, we make decisions at the bank granularity. A write request can be issued as a slow write only as long as there are no other operations (reads or writes) queued for the same bank. Figures 4 and 5 illustrate this scheme.

Figure 4 shows a situation with several read and write requests. For Bank 1, there is a single write request and no read requests. Therefore, the write request for Bank 1 can be issued as a slow write.

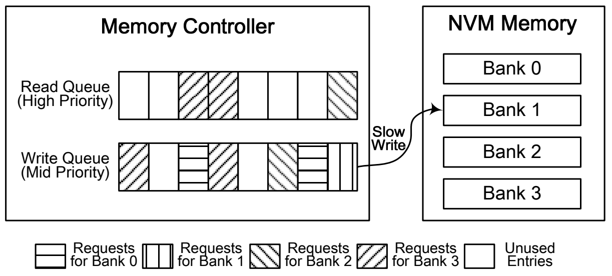

Figure 4. **Situation when *Bank-Aware Mellow Writes* scheme issues a slow write.**

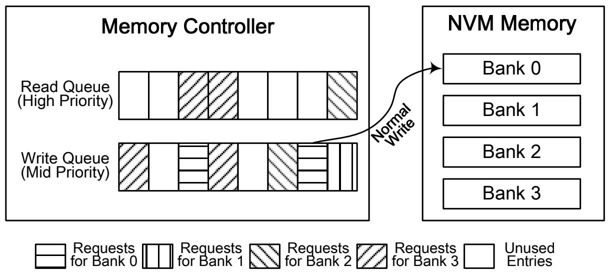

Figure 5. **Situation when *Bank-Aware Mellow Writes* scheme issues a normal write.**

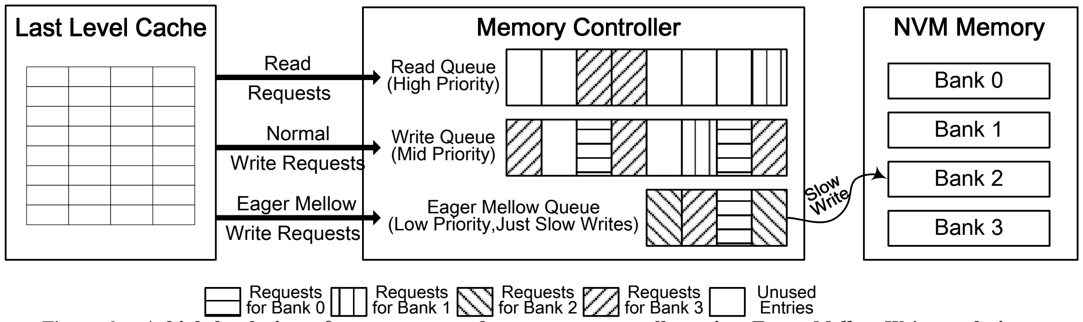

Figure 6. **A high level view of a processor and a memory controller using *Eager Mellow Writes* technique.**

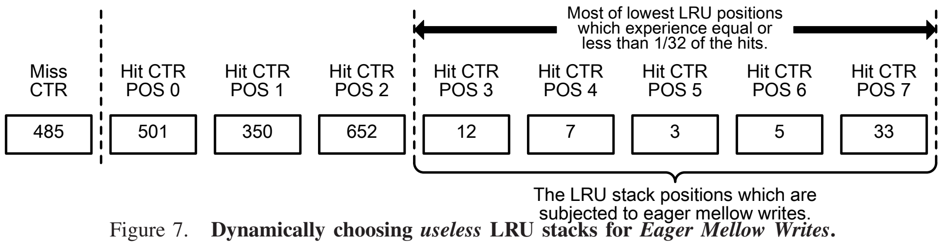
Most of lowest LRU positions which experience equal or less than 1/32 of the hits.
Miss CTR
485
Hit CTR POS 0
501
Hit CTR POS 1
350
Hit CTR POS 2
652
Hit CTR POS 3
12
Hit CTR POS 4
7
Hit CTR POS 5
3
Hit CTR POS 6
5
Hit CTR POS 7
33
The LRU stack positions which are subjected to eager mellow writes.

Figure 7. **Dynamically choosing *useless* LRU stacks for *Eager Mellow Writes*.**

Figure 5 shows a situation with two write requests for Bank 0 and no read requests for the same bank. In this case, the next write request for Bank 0 will be issued as normal write, since there is another write request waiting. This is to reduce the chances that the write queue will fill, triggering an expensive write drain.

Figures 4 and 5 also illustrate that no write requests can be issued to Banks 2 and 3, since there are read requests for the banks waiting. Read requests have higher priority than write requests.

One advantage of the *Bank-Aware Mellow Writes* technique is that it requires minimal changes to the memory controller. The only modifications are a mechanism to detect bank conflict in the read and write queues and implementation of the slow write technique. However, its glaring drawback is that, when there are no writes for a bank in the write queue, it is unable to take advantage of the bank idle time.

B. *Eager Mellow Writes*

Inspired by Lee *et al.'s* work [20], we introduce *Eager Mellow Writes* scheme, which takes advantage of bank idle time in the presence of no write requests. It allows the cache to eagerly write back some dirty data (that are predicted to not be used again) when the memory is not busy. In a nutshell, these items are placed into a third queue (*Eager Mellow Queue*). Items in *Eager Mellow Queue* have the least priority, can never trigger a write drain, and are only performed when there are no normal read or write requests to that bank.

Figure 6 shows a high level view of *Eager Mellow Writes* scheme. In this case, only slow write requests for *Bank 2* can be issued from *Eager Mellow Queue*. Requests for other banks cannot be issued from *Eager Mellow Queue* because there are outstanding requests for these banks in the write and/or read queues.

This scheme requires two changes. First, the LLC needs to identify what items to use as candidates for *Eager Mellow Writes*. These are sent to the memory controller. Second, the memory controller needs an additional *Eager Mellow Queue* to hold them. In the rest of this subsection, we will discuss in detail the design of *Eager Mellow Writes* technique.

1) *Identifying Eager Mellow Writes:*

Any cycle the LLC is idle and the *Eager Mellow Queue* is not full, the LLC has a chance to choose an item to be placed in the *Eager Mellow*

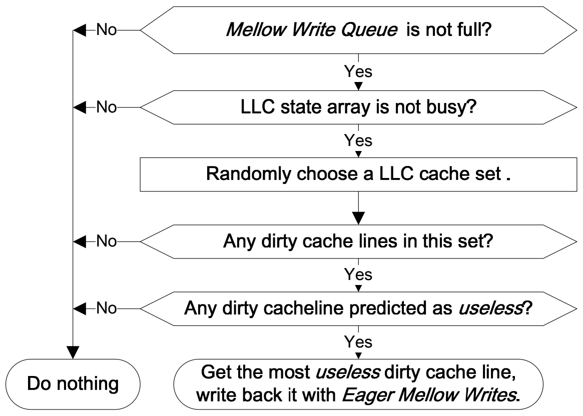
Mellow Write Queue is not full?
No
Yes
LLC state array is not busy?
No
Yes
Randomly choose a LLC cache set .
Any dirty cache lines in this set?
No
Yes
Any dirty cacheline predicted as useless?
No
Yes
Do nothing
Get the most useless dirty cache line, write back it with Eager Mellow Writes.

Figure 8. **Workflow of *Eager Mellow Writes* on LLC side.**

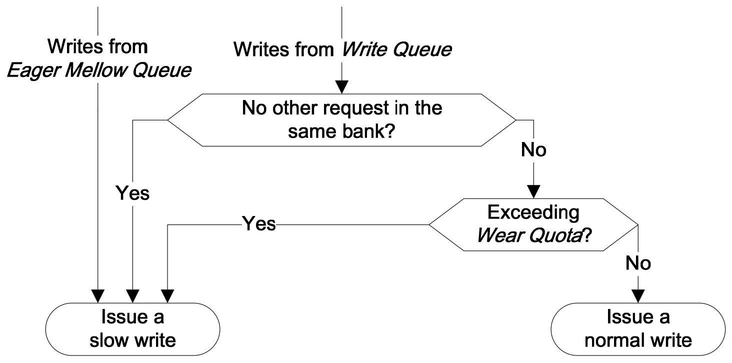

Figure 9. **How a memory controller using *Bank Aware Mellow Writes*, *Eager Mellow Writes*, and *Wear Quota* decides whether to use normal writes or slow writes.**

Queue. The goal is to find dirty lines in the cache that are unlikely to be modified again before they get evicted from the cache. If a line gets modified again before being evicted, then the write was wasted, reducing, rather than increasing, the lifetime.

Our scheme is as follows. The LLC randomly chooses a cache set. Within that set, it finds the dirty lines that are unlikely to be used again. If present, it issues an *Eager Mellow Write* of the least likely line to be used again. That line is marked clean, *not evicted*, from the cache.

The key design issue—how to find the useless dirty cache line, in a simple, energy-efficient, low-storage way? Given the stack property [21] of LRU replacement policy, and inspired by Qureshi *et al.*'s work [22][23], we propose a simple but effective scheme for an N-entry LRU stack (as shown in Figure 7):

- In the LLC controller, add a hit counter for every LRU stack position (Note: This is a single counter for the same LRU stack position across all sets, not per set). For an N-way associative LLC set, the *Most Recently Used* block is in LRU position 0, while *Least Recently Used* block is in LRU position (N-1). In addition, add a single miss counter to record the number of missed requests.
- On every LLC request (read or write), based on the hit stack position, increment the corresponding hit counter on a hit or miss counter on mis.
- After every T_{sample} period, we check the count of all the counters, identify the eager LRU position. This is the position such that the sum of the hits in eager LRU position through n-1 is less than *THRESHOLD_RATIO* of the requests. Since these higher LRU positions contribute so few hits, they are marked as *useless* until the next check and are subjected to *Eager Mellow Writes*. After we set the new *useless* LRU stack positions, we reset the hit counters and miss counter to 0, and restart the new round of profiling for the next T_{sample} period. In our experiment, T_{sample} is 500000 ns, and *THRESHOLD_RATIO* is $\frac{1}{32}$. As is shown in our motivational example of Figure 7, LRU positions 3—7 accumulate less than $\frac{1}{32}$ of the total LLC requests and therefore considered as *useless* and subjected to *Eager Mellow Writes*.

2) Performing Eager Mellow Writes: We add a third category of memory accesses to the memory controller. *Eager Mellow Writes* from the LLC are placed into the *Eager Mellow Queue*, which can only issue slow writes to banks. This queue has the lowest priority, no write drain operations, and is only issued when there are no same-bank requests in either of the read queue and the normal write queue. Also, to reduce the hardware overhead as well as reduce the number of *Eager Mellow Writes*, we use a relatively small *Eager Mellow Queue*—in our simulation, it has only 16 entries, whereas the read and write queues each have 32 entries.

C. *Wear Quota*

As will be shown in Section VI, although *Mellow Writes* greatly improve the lifetime of a resistive main memory system without noticeable performance penalty, with a memory-intensive workload, the lifetime can still fall below an acceptable threshold (e.g., 8 years). Here, we introduce another scheme, *Wear Quota*, to guarantee the lifetime for memory write intensive applications. The notion of *Wear Quota* is straight-forward: We divide the execution period into multiple sample periods (e.g., 500000ns per period in our experiment). If the accumulative wear of all previous periods surpasses their corresponding wear threshold, only slow writes can be issued in current period.

In our resistive memory system, the write operation is at a block granularity (64 bytes), and the endurance of each resistive memory block is $Endur_{blk}$ (in terms of normal writes). If we want the lifetime of a block to be at least $T_{lifetime}$, we can ensure its average wear during every period of T_{sample} to be at most $WearBound_{blk}$:

$$WearBound_{blk} = Endur_{blk} * \frac{T_{sample}}{T_{lifetime}}.$$

When referring to the lifetime of a memory bank, its wear in each sample period could be on average at most $WearBound_{bank}$:

$$WearBound_{bank} = BlkNum_{bank} * WearBound_{blk} * Ratio_{quota}$$

where $BlkNum_{bank}$ is the number of memory blocks of the memory bank, and $Ratio_{quota}$ is a number equal or smaller than 1.0. Ideally, $Ratio_{quota}$ should be 1.0. However, in our experiment (and also necessarily in real system), we guarantee even write distribution by using Start-Gap wear leveling technique. Since Start-Gap may introduce slightly extra wear, we conservatively set $Ratio_{quota}$ to be 0.9.

In the beginning of each period, the memory controller first calculates the value of $ExceedQuota$ of each memory bank:

$$ExceedQuota = \sum Wear_{bank} - WearBound_{bank} * Num_{previous\_periods}$$

where $\sum Wear_{bank}$ is total amount of wear placed on the memory bank, and $(WearBound_{bank} * Num_{previous\_periods})$ is the *Wear Quota* of all previous periods. If $ExceedQuota$ is larger than 0, it means total wear of the previous periods exceeds corresponding quota. **In this case, to reduce the average amount of wear per period, this memory bank can only perform slow writes in the coming period.**

D. *Put It All Together*

Our three proposed schemes (*Bank-Aware Mellow Writes*, *Eager Mellow Writes* and *Wear Quota*) work together to deliver high performance most of the time, but still guarantee a certain memory lifetime. Figure 9 shows how a memory controller with these three schemes decides when to issue a slow write: *For each bank*, look for a write to perform.

- If there is a single request in the *Write Queue*, issue a slow write.
- If there are multiple requests, but the *Wear Quota* is exceeded, issue a slow write.
- If there are multiple requests and the *Wear Quota* is not exceeded, issue a normal write.
- If there are no requests in the *Write Queue*, and there is a request in the *Eager Mellow Queue*, issue a slow write.

E. *Overhead Discussion*

In this subsection we will discuss the hardware and energy overhead of proposals.

- **Additional Voltage Supply.** Our proposed scheme requires a lower voltage supply to enable the slow write operation with smaller dissipated power. Since resistive memory circuits typically already require more than one voltage supply (e.g., write and read operations typically need different voltages), there is an affordable design overhead.
- **Storage Overhead.** The LLC, for *Eager Mellow Writes*, requires some additional storage—a cycle counter, a miss counter and number of LLC associativity hit counters, where each counter is $\lceil log_2 \frac{T_{sample}}{T_{proc\_clk}} \rceil$ bits. In our experiments, the LLC has an associativity of 16, T_{sample} is 500000ns, and T_{proc_clk} (processor clock period) is 0.5 ns. The total additional storage in LLC is $\lceil log_2 \frac{T_{sample}}{T_{proc\_clk}} \rceil * (1 + 1 + ASSOC_{LLC}) = 20 * 18 = 360$ bits. The memory controller also requires additional storage. *Eager Mellow Writes* requires a 16-entry queue for each memory channel, and *Write Quota* requires three registers (64-bit each) in each bank to record the total number of normal writes, slow writes and periods to the corresponding memory bank.
- **Energy Overhead.** Additional energy is used in the LLC and memory. When finding useless dirty cache lines, the LLC state array is the only RAM needed to be accessed; LLC tag/data RAMs will be accessed only when a useless dirty cache line is found and needed to be issued as a mellow eager writeback. Our technique introduces extra energy consumption in memory because (1) *Eager Mellow Writes* and *Write Cancellation* generate extra number of writes, and (2) a slow write consumes more energy than a normal write. However, in Section VI we will quantitatively show that the additional memory-side energy consumption is moderate compared with whole system energy.

Overall, compared with the lifetime benefit, our design requires minimal hardware overhead and a moderate increase in main memory energy consumption.

V. Methodology

We use the gem5 simulator [25] with NVMain [26], a timing-accurate main memory simulator for non-volatile memory technologies. Table I provides the processor and cache details, and Table II provides the memory system details. For a given workload, we assume the system will cyclically execute the same execution pattern. Then the lifetime is calculated as how much time it takes until one cell in the memory system reaches its wear limit.

Without loss of generality, we model ReRAM technologies [27] which have a representative *Expo_Factor* of 2.0. ReRAM represents a wide range of technologies, with their write latency ranging from few nanoseconds [28] to millisecond scale [29], and their endurance ranging from few hundreds [30] to 10^{12} scale [31]. Here we consider representative memory-grade ReRAM devices with 150ns normal write latency and $5 * 10^6$ normal write endurance. Recent commercial efforts appear to also be in line with our baseline [32][33].

We simulate multiple memory write policies, as shown in Table III. There are several configurations that can be

Table I
PROCESSOR SIMULATION PARAMETERS

Freq.	2GHz
Core	OoO, Alpha ISA, 8-issue, 64-byte cacheline
L1$	split 32KB I/D-caches, 4-way, 2-cycle hit latency, 8-MSHR
L2$	256KB, 8-way, 12-cycle hit latency, 12-MSHR
L3$ (LLC)	2MB, 16-way, 35-cycle hit latency, 32-MSHR, Useless threshold is $\frac{1}{32}$, profiling period is 500,000ns.

Table II
MAIN MEMORY SYSTEM SIMULATION PARAMETERS

Basics	400 MHz, 64-bit bus width, using ReRAM, using Start-Gap wear-leveling [24] in bank granularity, write-through (writes bypass row buffers), 1KB row buffer, open page policy, tFAW=50ns
# of Banks	Three options: –4 banks, distributed in 1 ranks –8 banks, distributed in 2 ranks –16 banks, distributed in 4 ranks (default)
Read Queue	32 entries, highest priority
Write Queue	32 entries, middle priority, write drain threshold: 16 (low), 32 (high)
Eager Mellow Write Queue	16 entries, lowest priority, no write drain, slow writes
Wear Quota Parameters	Expected Lifetime: 8 Years *Wear Quota* sample period: 500,000ns *Wear Quota* threshold ratio ($Ratio_{quota}$ in Section IV): 0.90
Row Size	16KB
tRCD	48 cycles (120 ns)
tWP (wr. pulse time)	normal writes: 60 cycles (150 ns); 1.5x slow writes: 90 cycles (225 ns); 2.0x slow writes: 120 cycles (300 ns); 3.0x slow writes (**default**): 180 cycles (450 ns).
tCAS	1 cycle (2.5 ns)
endurance	normal writes: $5.000 * 10^6$ writes; 1.5x slow writes: $1.125 * 10^7$ writes; 2.0x slow writes: $2.000 * 10^7$ writes; 3.0x slow writes (**default**): $4.500 * 10^7$ writes.

mixed and matched— normal vs slow writes, eager write-backs, bank-aware mellow write-backs, write cancellation involving normal and/or slow writes, and wear quota. *BE-Mellow+SC+NC* means a system with both *Bank-Aware* and *Eager Mellow Writes*, and both normal writes and slow writes are cancellable; *BE-Mellow+SC+WQ* is the same except that only slow writes are cancellable and *Wear Quota* scheme is used. If without specific mentioning, we use default slow writes with 3.0× latency in all the write policies.

In order to reduce the number of results displayed, we have chosen configurations that result in the best performance for the general option. For example, we show Norm and E-Norm+NC, but neither Norm+NC, nor E-Norm. This is because normal writes do not benefit enough from write cancellation to justify the drop in endurance. Write cancellation is important for eager write performance because it can avoid eager writes blocking the incoming reads. Also, the eager write queue does not trigger write drains, so cancelling eager slow writes will not increase the possibility of write drains.

We use nine memory-intensive benchmarks from SPEC2006. To test the performance of our schemes under random and stream memory access patterns, we also include `GUPS` and `stream` benchmarks. We list these benchmarks in Table IV with their MPKI (miss per 1000 Instructions).

Table III
MEMORY WRITE POLICIES

Basic Policies	
Norm	Just using normal writes
Slow	Just using slow writes
B-Mellow	Using *Bank-Aware Mellow Writes*
BE-Mellow	Using both *Bank-Aware* and *Eager Mellow Writes*
E-Norm	Just using normal writes, but with eager writes
E-Slow	Just using slow writes, but with eager writes
Additional Write Choices	
+NC	Normal writes are cancellable
+SC	Slow writes are cancellable
+WQ	With *Wear Quota* scheme

Table IV
WORKLOADS AND THEIR MPKI (MISS PER 1000 INSTRUCTIONS) WITH A 2MB LLC

Workload	MPKI	Workload	MPKI	Workload	MPKI
leslie3d	5.95	hmmer	1.34	milc	19.49
GemsFDTD	15.34	zeusmp	4.53	mcf	56.34
libquantum	30.12	bwaves	5.58	lbm	31.72
stream	12.28	gups	8.91		

We warm up the cache for 6 billion instructions and then simulate in detail for another 2 billion instructions.

VI. RESULTS

In order to evaluate our *Mellow Writes* schemes, we first present the main tradeoff—performance vs lifetime. In order to better understand the reasons for these results, we break the performance down into bank utilization, write drain time, memory requests from the LLC, and memory requests issued to the memory banks. We also report the energy consumptions of the main memory system. We then provide a sensitivity study of several key parameters. Finally, we compare our best *Mellow Writes* with various kinds of static mechanisms. When not specifically stated, we use 3x write latency for all the slow writes.

A. Performance vs. Lifetime

We begin by presenting the fundamental tradeoff in our system: performance vs. lifetime. Figures 10 and 11 show the performance and lifetime, respectively, of our applications. We make the following observations:

- *E-Norm+NC* is designed for the highest performance. However, although it indeed gets the best performance for most of the benchmarks, it performs considerably worse than default (*Norm*) for `lbm` (11% lower IPC) and also performs worse than *BE-Mellow+SC+WQ* for `libquantum`. This is because write cancellation may increase the write drain possibility by letting writes stay in write queue longer. Also *E-Norm+NC* has an unacceptable short lifetime. The lifetime suffers because of more write requests caused by eager writebacks and write cancellation. Therefore, eager writes and write cancellation should not be adopted with normal writes.
- *E-Slow+SC* has the longest lifetime. Unfortunately, the latency is far too high (geometric mean: 0.77x performance), with the worst being 0.46x (*lbm*). Even with write cancellation, implementing a system with only slow writes is not feasible.

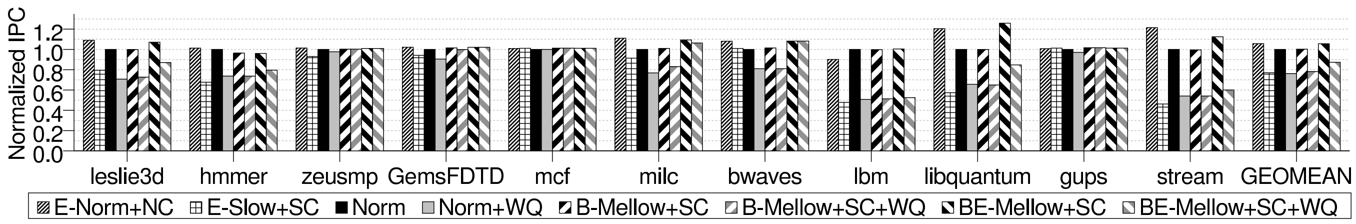

Figure 10. **IPC (instruction per cycle) of systems with different write policies.**

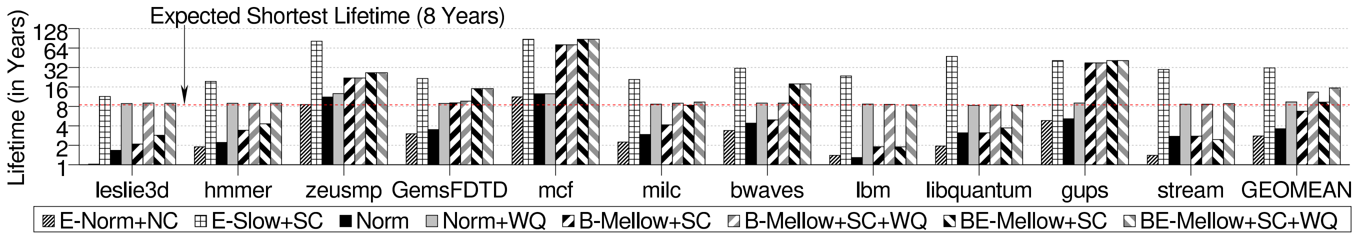

Figure 11. **Resistive Memory Lifetime (in years, log scale) of systems with different write policies.**

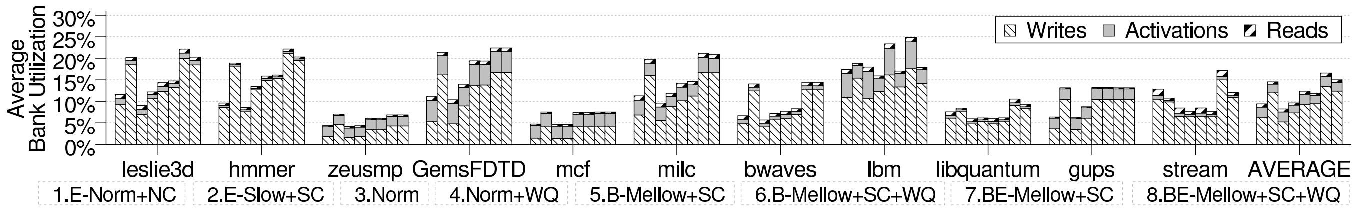

Figure 12. **Average bank utilization of systems with different write policies.**

- Using *Bank-Aware Mellow Writes* (*B-Mellow+SC*) improves the lifetime with negligible loss in performance.
- Combining *Bank-Aware Mellow Writes* with *Eager Mellow Writes* (*BE-Mellow+SC*) results in better performance and better lifetime than just using *Bank-Aware Mellow Writes* (*B-Mellow+SC*) . Overall, *BE-Mellow+SC* is a nice balance between lifetime and performance (Lifetime: 9.30 years; IPC: 1.06x of *Norm*).
- Although *Mellow Writes* schemes greatly improve the lifetime on average, some benchmarks achieve below the shortest acceptable liftime (e.g., 8 years). In these cases, the *Wear Quota* scheme raises the lifetime to at least 8 years (see *+WQ* items in Figure 11). Not surprisingly, *Wear Quota* comes with performance cost. However, if we look at the three configurations with the Wear Quota, *BE-Mellow+SC+WQ* achieves the best performance. This is because, if the wear quota is reached, all schemes will have the same number of slow and normal writes, but the *Mellow Write* schemes will make better decisions about *which* writes should be slow vs normal. Therefore, for *Mellow Write* schemes, fewer sample periods are needed to use all slow writes than the normal scheme.

B. Bank Utilization

The utilization of a bank refers to the percentage of the time when corresponding bank is busy. We can see in Figure 12 that, not surprisingly, all configurations utilizing slow writes result in higher bank utilization. The mellow writes techniques (*B-Mellow+SC*, *E-Mellow+SC* and *BE-Mellow+SC*) sometimes (e.g., `lbm`) result in higher utilization than globally slow writes with eager writes (*E-Slow+SC*), which may appear counter-intuitive. The reason for this phenomenon is that *E-Slow+SC* has considerably worse performance (e.g., `lbm`) than mellow writes techniques, therefore fewer requests are sent to the memory controller in the same time period.

Table V
ReRAM CELL PARAMETERS

		Read	Norm Set	Slow Set	Norm Reset	Slow Reset
Voltage (V)		0.20	1.00	0.95	1.00	0.95
Latency (nv)		–	150	450	150	450
Power (uW)		0.02	–	–	–	–
Energy per cell (pJ)	CellA	–	0.1	0.23	0.1	0.23
	CellB	–	0.2	0.46	0.2	0.46
	CellC	**–**	**0.4**	**0.92**	**0.4**	**0.92**
	CellD	–	0.8	1.84	0.8	1.84
	CellE	–	1.6	3.68	1.6	3.68

Table VI
ENERGY PER OPERATION OF MEMRISTIVE MAIN MEMORY.

	Buffer Read (pJ)	Norm Write (pJ)	Slow Write (pJ)	Slow-Norm Write Energy Ratio
CellA	1503.0	248.8	314.5	1.26
CellB	1503.0	300.0	432.3	1.44
CellC	**1503.0**	**402.4**	**667.8**	**1.66**
CellD	1503.0	607.2	1138.8	1.88
CellE	1503.0	1016.8	2080.9	2.05

C. Write Drain Time

Write drains refer to the situation when the write queue occupancy reaches a threshold (usually full). When this occurs, the system prioritizes writes over reads until the write queue is drained. This is an expensive memory operation that directly impacts performance by delaying time-critical reads. This is also the main drawback of using slow writes:

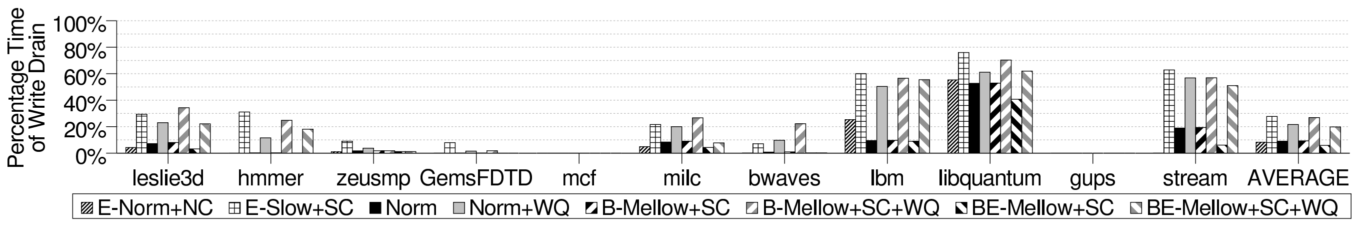

Figure 13. **Percentage of time used by write drain operations.**

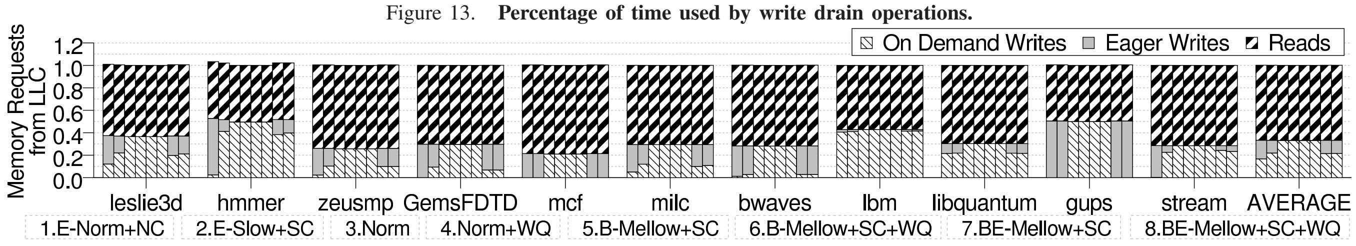

Figure 14. **Number of memory requests from LLC to memory controller (normalized to the number of *Norm* policy).**

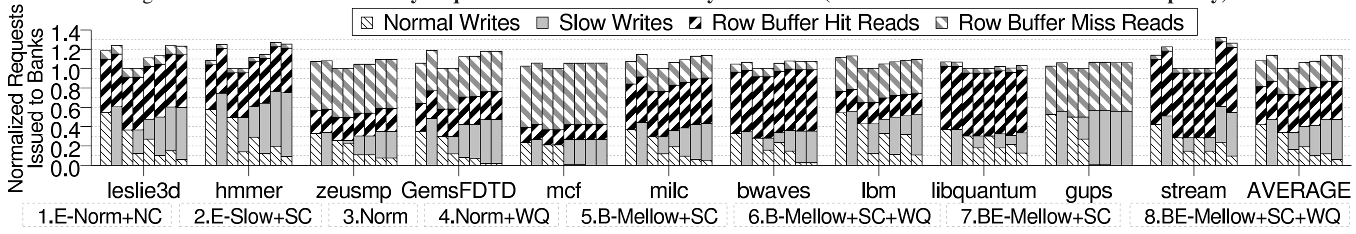

Figure 15. **Number of memory requests issued to memory banks (normalized to the number of *Norm* policy).**

higher queue occupancy leading to more write drains. We study in detail of the write drain time in Figure 13.

When globally using the slow writes, the write queue fills often even if the *Eager Writes* are used (*E-Slow+SC*). *Bank Aware Mellow Writes* does not increase write drains (compared with normal), and *Eager Mellow Writes* (*BE-Mellow+SC*) limit the write drain time to within 6% of the total execution time by proactively writing back data when the system is not busy. It is not surprising that using the *Wear Quota* scheme will increase the possibility of write drains. However, the percentages of write drain time in configurations with *Wear Quota* (*Norm+WQ*, *B-Mellow+SC+WQ* and *BE-Mellow+SC+WQ*) are still smaller than the write drain percentage when globally using slow writes (*E-Slow+SC*).

D. Memory Requests from LLC

We can see from Figure 14 that using *Eager Writes* transforms, on average, nearly half of the writes from normal writes to eager writes. Although *Eager Writes* may increase the number of write memory requests because of the inaccurate prediction of the unused blocks, this phenomenon is not obvious in our experiment (up to 2.2% increase of writes in benchmark `hmmer`). This reflects the fact that our mechanism to identify useless dirty blocks (described in Section IV-B1) is relatively accurate.

E. Memory Requests to Memory Banks

Figure 15 shows the effects of write cancellation (which results in a second write after the read completes) and eager writebacks (which are wasted if the cacheline is modified again before eviction) on the number of issued memory requests to banks. *BE-Mellow+SC* issues substantially more requests to memory banks than *Norm*. However, as shown in Figure 14, the additional write requests due to eager write-backs are relatively few. Therefore, it is write cancellation which is mainly responsible for the increase of the issued writes to bank.

F. Energy Consumption of Main Memory

A potential drawback of a slow write is that it may consume more energy than a normal write. Therefore we simulate in detail the energy consumption of our schemes.

The detailed energy simulation parameters are shown in Table V. We use ReRAM under 22nm process. We assume that a 3× slow write comes with 0.767× dissipated power of a normal write, due to exponential dependence of ionic velocity on temperature [15]. Therefore, a slow write to the same ReRAM cell consumes 2.3× energy of a normal write. Since the set/reset energy of a cell is a crucial design point of ReRAM, we model five different cells with their normal set/reset energy from 0.1pJ (*CellA*) to 1.6pJ (*CellE*). Then we use nvsim [34] to obtain the energy consumption of buffer read (in row buffer granularity) and normal/slow write (in cacheline granularity) operations of resistive main memory. We assume half of the bits in a write operation are subjected to the Set operation, and the other ones are subjected to Reset. We can see from Table VI that, as the cell write energy decreases, the energy consumption difference of normal and slow writes also decreases—for *CellE* (1.6pJ/cell for normal set/reset), a slow write takes 2.05× energy of a normal write; while for *CellA* (0.1pJ/cell

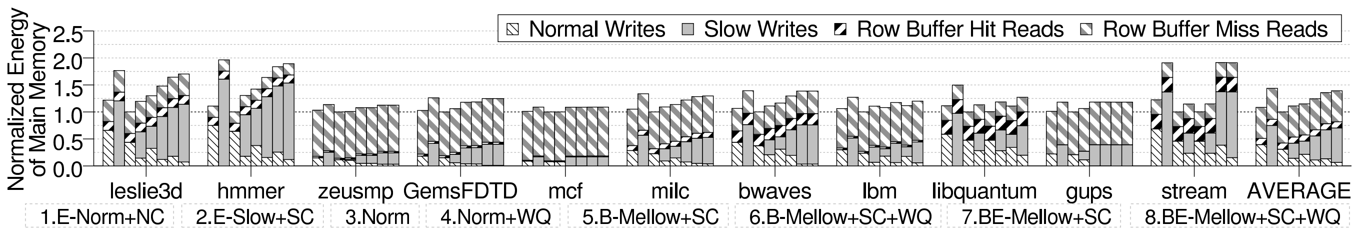

Figure 16. **Energy consumption of main memory (see *CellC* in Table VI, normalized to *Norm*).**

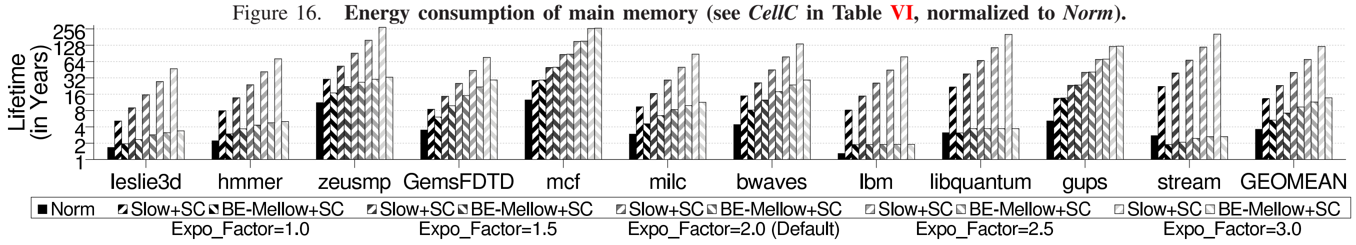

Figure 17. **Sensitivity to exponent of latency to lifetime improvement relationship Equation 2.**

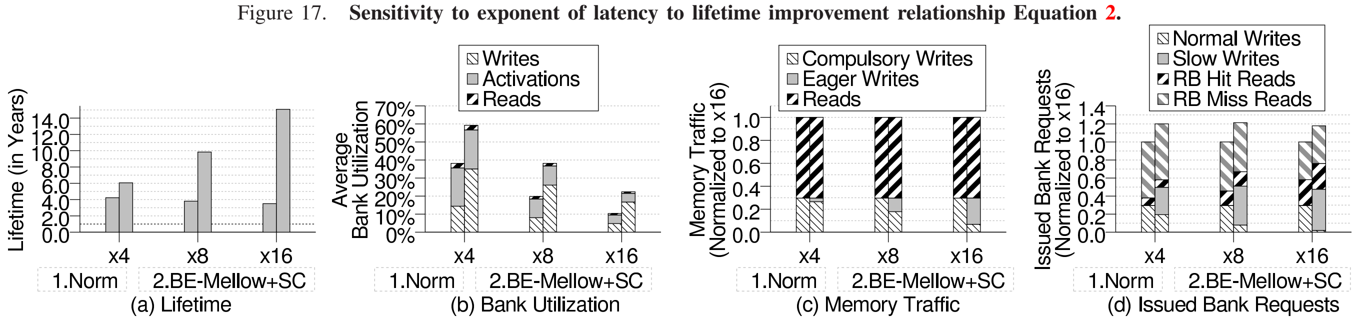

Figure 18. **Sensitivity to bank-level parallelism of benchmark GemsFDTD.**

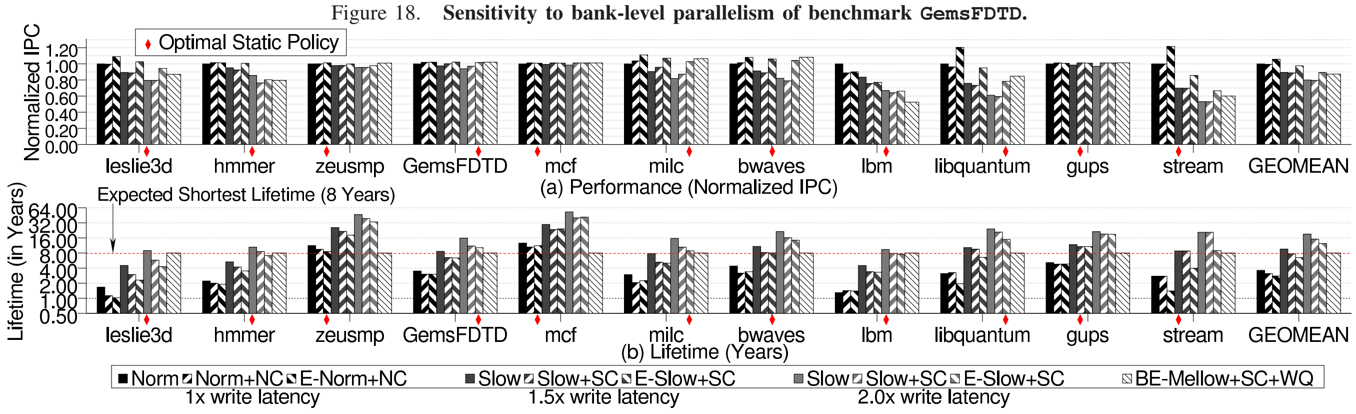

Figure 19. **Comparing *BE-Mellow+SC+WQ* with various kinds of static policies.** For each benchmark, the column with red diamond is best static mechanism (i.e., the static mechanism which guarantees 8 years lifetime and delivers the best perfomance). We can see that there is no single static mechanism which suits all the benchmarks, and our *BE-Mellow+SC+WQ* outperforms or equals to the best static mechanism in 8 out 11 benchmarks.

for normal set/reset), a slow write only takes 1.26× energy of a normal write. This is because the perpheral circuit also consumes energy while writing, when cell write energy decreases the energy consumption of perpheral circuit becomes more dominant.

We then calculate the whole main memory energy using the numbers from *CellC* in Table VI and assume energy of a row-buffer hit read is 100pJ. The results are shown in Figure 16. On average, our best configuration (*BE-Mellow+SC+WQ*) consumes around 0.39× more energy in main memory system than the default configuration (*Norm*). Given the fact that the main memory system usually consumes a relatively small portion of the whole system energy, the energy increase of *Mellow Writes* schemes is moderate/trivial compared to the total system energy consumption.

G. Sensitivity to the Analytic Model

In Section II, we show that if the write latency is slowed down by N times, an N^{Expo_Factor} times endurance can be achieved. And for resistive memory, an *Expo_Factor* of 2 is reasonable for our exploration. However, for various kind of non-volatile memory technologies, their *Expo_Factors* might be different. We investigate how our system performs as the exponential relationship between latency and lifetime changes. In addition to our default 2.0, we compare the lifetime of our schemes with four other *Expo_Factor* values, which are 1.0, 1.5, 2.5 and 3.0. We want to find out how dependent our results are on this exponential factor.

Figure 17 shows the results. Not surprisingly, the lifetime of both *Slow+SC* and *BE-Mellow+SC* improves when *Expo_Factor* goes up. However, the lifetime increase of *BE-Mellow+SC* is not as dramatic as for *Slow+SC*. The former gets around 0.5× more lifetime for an *Expo_Factor* of 3.0 instead of 2.0, whereas the latter gets around 2× more. This is because Mellow Writes schemes issue some amount of normal writes, and these normal writes contribute to a fixed amount of wear no matter how large *Expo_Factor* is. An important discovery is that, even for an *Expo_Factor* as pessimistic as 1.0, *BE-Mellow+SC* still gets a lifetime which is 1.47× of the lifetime in a baseline system (*Norm*). Therefore, *Mellow Writes are useful for a range of exponents, and thus technologies, and do not depend on a superlinear relationship between latency and endurance benefit*.

H. Sensitivity to Bank-Level Parallelism

Here we demonstrate how available bank-level parallelism affects performance of *Mellow Writes*. Figure 18 shows the behavior of benchmark `GemsFDTD` with different numbers of banks. We can see from Figure 18 (a) that, as the number of banks decreases, the lifetime difference between *Norm* and *BE-Mellow+SC* diminishes, indicating the effectiveness of *Mellow Writes* diminishes. It is not surprising that this affects Bank-Aware mechanisms, since they depend on asymmetric use of banks. Figure 18(b) reveals the reason—as bank-level parallelism decreases, the utilization of each bank increases, leaving fewer intervals for eager and slow writes. Figure 18(c) clearly shows that the number of eager writes decreases dramatically as the number of banks decreases. Also, Figure 18(d) shows that the number of issued normal writes to the banks increases substantially as the number of banks decreases. Note that, although a substantial number of slow writes are issued with 4 banks, most of them get cancelled before finishing due to incoming reads.

Therefore, to ensure the effectiveness of *Mellow Writes*, it is important to guarantee a sufficient number of banks.

I. Mellow Writes vs Static Policies

In Section III, we show that it is hard to have a static mechanism (i.e., with fixed write latency and write policy) to fit different applications. To show the effectiveness of our Mellow Writes Policy, we compare *BE-MELLOW+SC+WQ* (our best Mellow Writes policy which guarantees a minimal lifetime) with various kinds of static mechanisms. Since our modified Eager Writes policy can also be applied to a system with static write latency, we also include these static mechanisms (*E-Norm+NC* and *E-Slow+SC*) in our evaluation. We shown the results in Figure 19. For each benchmark, the column with red diamond is the best static policy (the one guaranteeing minimal lifetime and delivering the best performance). We can see that, the best static mechanism is different for different benchmark, thus there is no single static mechanism which fits all applications.

As is shown in Figure 19, *BE-MELLOW+SC+WQ* successfully guarantees the minimal lifetime (in our experiment, 8 years) in all the applications. In 8 out of 11 applications, *BE-MELLOW+SC+WQ* outperforms or at equals the best static mechanism. We also investigate why our mechanism delivers worse performance in the rest of the benchmarks (i.e., `hmmer`, `lbm` and `stream`)–it turns out these benchmarks are very sensitive to write latency in some cases. A possible modification for this situation is to adopt multiple write latencies instead of just two in our schemes. In this case, deciding which write latency to use is a major challenge, and this will be our future work.

Overall, *BE-MELLOW+SC+WQ* shows a nice balance between lifetime and performance requirements for all the benchmarks, and such a balance cannot be fulfilled with a single static mechanism.

VII. Related Work

Wear Leveling and Limiting. Many techniques exist for implementing wear-leveling for non-volatile memories. Start-Gap [24] employs a novel shift-based approach that is able to achieve 95% of ideal memory lifetime with only 8 bytes of storage overhead. We use Start-Gap in our system. Other shift-based approaches exist, such as shifting cache lines with a page [1], and shifting bits in a line, or lines in a segment [35]. Security Refresh [36] uses randomized address mappings within a bank for distribute wear as well as prevent malicious wear-out attacks. Well-known techniques exist for wear-limiting, such as DRAM buffering [1]. Others, such as Flip-N-Write [2], exploit specific properties of the data being written to reduce the number of writes on a per-cell basis. Similar to DRAM buffering, Saadeldeen *et al.* [37] also use a small SRAM buffer in their ReRAM-based branch prediction scheme. All such techniques can be classified as *physical* techniques, in that they alter the actual contents of the memory in order to reduce wear. Our mellow writes concept is orthogonal to these, as it uses *temporal* properties of write operations to reduce wear.

Latency-Density Tradeoffs in MLC NVMs. It is well know that, when using MLC (multi-level cell) NVM, there is a trade-off between write/read latency and storage density. Prior proposals try to balance such a trade-off in main memory [38][39], file storage [40] and coherence directories [41] by adptively using fast and slow operations. The basic notion of these proposals is to use fast(single-level) accesses for performance critical read/writes operations, and slow(multi-level) accesses for other ones. In this work, we also follow this intuition to avoid performance loss by not using slow writes in performance-critical situations.

Write Cancellation and Eager Writeback. To avoid of performance loss due to long latency write in NVM, Qureshi *et al.* [18] introduce the concept of *write cancellation* (also known as *read premption* [19]) which services the incoming reads immediately by cancelling conflict writes. Lee *et al.* [20] propose a method by which LRU dirty cache lines are written back before eviction, called *eager writeback*. Doing so reduces memory bandwidth contention between demand reads and writebacks, thus improving system performance. Qureshi *et al.* [42] also propose a scheme to improve PCM performance by early and eagerly writing back the long latency SET operations, which can be viewed as a variation of *eager writeback*. Both write cancellation and a modified version of eager writeback are integral parts of our mellow writes concept.

Cache Management. Qureshi *et al.* [22] propose a method of partitioning an LRU-policy cache among concurrently executing applications. This work provides the motivation for our *Eager Mellow Writes* writeback criteria, in that we identify writeback candidates based on their utility rather than simply their LRU stack position. Some other cache management frameworks may also work with *Eager Mellow Writes*. In particular, we are interested in *Dead Block Prediction* methods [43][44], since these proposals directly predict which cache block are no longer used and thus can be eagerly and slowly written back. We believe that by using *Dead Block Prediction*, we can further improve the effectiveness of *Eager Mellow Writes*.

VIII. Conclusions and Future Work

In this paper we explore the trade-off that, for non-volatile memories, there exists a linear to cubic (representatively to be quadratic for resistive memories) endurance advantage to performing writes slowly using a smaller write dissipated power. Although slower writes can dramatically improve lifetime, they may also degrade overall system performance.

To utilize the lifetime benefit of slow writes as well as avoid their performance penalty, we introduce two schemes that issue slow writes when the memory system is not busy, namely *Bank-Aware Mellow Writes* and *Eager Mellow Writes*. The *Bank-Aware Mellow Writes* scheme performs slow writes only when there is no other request to the same bank. The *Eager Mellow Writes* scheme eagerly and slowly writes back the dirty blocks in the LLC which are predicted to have a low probability of being re-written. In addition, we also introduce *Wear Quota* scheme to ensure the minimal expected memory lifetime—when the *Wear Quota* of bank in all previous sample periods is reached, the bank can only issue slow writes in the coming period.

Our experiments show that a combination of two *Mellow Writes* schemes extends lifetime with minimal performance impact. This combination can achieve 2.58× in lifetime and 1.06× in performance of a baseline system using normal write speed. Meanwhile, *Wear Quota* is a safe scheme which can guarantee the minimal expected lifetime (e.g., 8 years) with relatively small performance loss.

We plan to continue exploring different aspects of the design space of *Mellow Writes*. First, we believe the effectiveness of *Eager Mellow Writes* schemes can be further improved by more accurately detecting useless cache blocks. Second, we are working on architectural support of *Mellow Writes* on NAND Flash technologies. Finally, we are also interested in OS-level support (e.g., by using frameworks similar with Liu et al.'s work [45][46]) for *Mellow Writes*.

IX. Acknowledgements

We would like to thank Moin Qureshi for his generous feedback in the initial phase of this project. This work was supported in part by NSF grants CCF-1017578/1528205/1461698/1500848/1533933 and DOE grant DE-SC0013553. We acknowledge support from the University of Chicago Research Computing Center, UCSB Center for Scientific Computing at CNSI and MRL (supported by NSF MRSEC DMR-1121053 and Hewlett Packard).

References

[1] 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*, 2009, pp. 24–33.

[2] 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*, 2009, pp. 347–357.

[3] D. B. Strukov, "Endurance-write-speed tradeoffs in non-volatile memories," *Applied Physics A*, vol. 122, no. 4, pp. 1–4, 2016.

[4] X. Liu *et al.*, "High-quality aluminum-oxide tunnel barriers for scalable, floating-gate random-access memories (fgram)," in *Proc. Int. Conf. on Memory Technology and Design (ICMTD)*, 2007, pp. 235–237.

[5] H.-C. Yu *et al.*, "Cycling endurance optimization scheme for 1mb stt-mram in 40nm technology," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, 2013, pp. 224–225.

[6] K. K. Likharev, "Layered tunnel barriers for nonvolatile memory devices," *Applied Physics Letters*, vol. 73, no. 15, pp. 2137–2139, 1998.

[7] M. D. Pickett *et al.*, "Switching dynamics in titanium dioxide memristive devices," *Journal of Applied Physics*, vol. 106, no. 7, p. 074508, 2009.

[8] J. McPherson *et al.*, "Thermochemical description of dielectric breakdown in high dielectric constant materials," *Applied Physics Letters*, vol. 82, no. 13, pp. 2121–2123, 2003.

[9] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnology*, vol. 8, no. 1, pp. 13–24, 2013.

[10] K. Likharev, "Electronics below 10 nm," in *In Nano and Giga Challenges in Microelectronics*, 2003, pp. 27–68.

[11] V. G. Karpov *et al.*, "Field-induced nucleation in phase change memory," *Phys. Rev. B*, vol. 78, no. 5, p. 052201, 2008.

[12] S. Raoux *et al.*, "Phase change materials," *MRS bulletin*, vol. 37, no. 02, pp. 118–123, 2012.

[13] E. Tsymbal *et al.*, "Ferroelectric and multiferroic tunnel junctions," *MRS bulletin*, vol. 37, no. 2, pp. 138–143, 2012.

[14] R. Waser *et al.*, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced Materials*, vol. 21, no. 25-26, pp. 2632–2663, 2009.

[15] D. B. Strukov and R. S. Williams, "Exponential ionic drift: fast switching and low volatility of thin-film memristors," *Applied Physics A*, vol. 94, no. 3, pp. 515–519, 2009.

[16] N. Mott and R. Gurney, *Electronic Processes in Ionic Crystals*, 2nd ed. Dover, New York, 1940.

[17] N. F. Mott and R. W. Gurney, "Electronic processes in ionic crystals," 1948.

[18] M. Qureshi, M. Franceschini, and L. Lastras-Montano, "Improving read performance of phase change memories via write cancellation and write pausing," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, 2010, pp. 1–11.

[19] G. Sun *et al.*, "A novel architecture of the 3d stacked mram l2 cache for cmps," in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, 2009, pp. 239–249.

[20] H.-H. S. Lee, G. S. Tyson, and M. K. Farrens, "Eager writeback - a technique for improving bandwidth utilization," in *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, 2000, pp. 11–21.

[21] R. L. Mattson *et al.*, "Evaluation techniques for storage hierarchies," *IBM Systems journal*, vol. 9, no. 2, pp. 78–117, 1970.

[22] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 423–432.

[23] M. K. Qureshi *et al.*, "Adaptive insertion policies for high performance caching," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, 2007, pp. 381–391.

[24] ——, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 14–23.

[25] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.

[26] M. Poremba and Y. Xie, "Nvmain: An architectural-level main memory simulator for emerging non-volatile memories," in *VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on*, 2012, pp. 392–397.

[27] C. Xu *et al.*, "Overcoming the challenges of crossbar resistive memory architectures," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, Feb 2015, pp. 476–488.

[28] J. Choi *et al.*, "Different resistance switching behaviors of nio thin films deposited on pt and srruo3 electrodes," *Applied Physics Letters*, vol. 95, no. 2, p. 022109, 2009.

[29] I. H. Inoue *et al.*, "Nonpolar resistance switching of metal/binary-transition-metal oxides/metal sandwiches: Homogeneous/inhomogeneous transition of current distribution," *Physical Review B*, vol. 77, no. 3, p. 035105, 2008.

[30] R. Waser *et al.*, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced Materials*, no. 21, pp. 2632–2663, 2009.

[31] M.-J. Lee *et al.*, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric ta2o5- x/tao2-x bilayer structures," *Nature materials*, vol. 10, no. 8, pp. 625–630, 2011.

[32] "Intel and Micron produce breakthrough memory technology," https://newsroom.intel.com/news-releases/intel-and-micron-produce-breakthrough-memory-technology/.

[33] "Sandisk and HP launch partnership to create memory-driven computing solutions," https://www.sandisk.com/about/media-center/press-releases/2015/sandisk-and-hp-launch-partnership.

[34] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 7, pp. 994–1007, 2012.

[35] P. Zhou *et al.*, "A durable and energy efficient main memory using phase change memory technology," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, 2009, pp. 14–23.

[36] N. H. Seong, D. H. Woo, and H.-H. S. Lee, "Security refresh: Prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, 2010, pp. 383–394.

[37] H. Saadeldeen *et al.*, "Memristors for neural branch prediction: a case study in strict latency and write endurance challenges," in *Proceedings of the ACM International Conference on Computing Frontiers*, 2013, pp. 26:1–26:10.

[38] M. K. Qureshi *et al.*, "Morphable memory system: A robust architecture for exploiting multi-level phase change memories," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, 2010, pp. 153–162.

[39] Z. Deng *et al.*, "Herniated hash tables: Exploiting multi-level phase change memory for in-place data expansion," in *Proceedings of the 2015 International Symposium on Memory Systems*, 2015, pp. 247–257.

[40] X. Dong and Y. Xie, "Adams: Adaptive mlc/slc phase-change memory design for file storage," in *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, 2011, pp. 31–36.

[41] L. Zhang *et al.*, "Spongedirectory: Flexible sparse directories utilizing multi-level memristors," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, 2014, pp. 61–74.

[42] M. K. Qureshi *et al.*, "Preset: Improving performance of phase change memories by exploiting asymmetry in write times," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, 2012, pp. 380–391.

[43] A.-C. Lai, C. Fide, and B. Falsafi, "Dead-block prediction & dead-block correlating prefetchers," in *Proceedings of the 28th Annual International Symposium on Computer Architecture*, 2001, pp. 144–154.

[44] H. Liu *et al.*, "Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency," in *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 222–233.

[45] L. Liu *et al.*, "A software memory partition approach for eliminating bank-level interference in multicore systems," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, 2012, pp. 367–376.

[46] ——, "Going vertical in memory management: Handling multiplicity by multi-policy," in *Proceedings of the 41st International Symposium on Computer Architecture*, 2014, pp. 169–180.