5-2009

Pairing Software-Managed Caching with Decay Techniques to Balance Reliability and Static Power in Next-Generation Caches

Kelly Shaw
University of Richmond, kshaw@richmond.edu

Margaret Martonosi

Recommended Citation

Shaw, Kelly and Margaret Martonosi. *Pairing Software-Managed Caching with Decay Techniques to Balance Reliability and Static Power in Next-Generation Caches*. Technical paper (TR-09-01). *Math and Computer Science Technical Report Series*. Richmond, Virginia: Department of Mathematics and Computer Science, University of Richmond, May, 2009.

University of Richmond Math and Computer Science Technical Report TR-09-01

Pairing Software-Managed Caching with Decay Techniques to Balance Reliability and Static Power in Next-Generation Caches

Kelly A. Shaw
Dept. of Math and Computer Science
University of Richmond
kshaw@richmond.edu

Margaret Martonosi
Dept. of Electrical Engineering
Princeton University
mrm@princeton.edu

Abstract

Since array structures represent well over half the area and transistors on-chip, maintaining their ability to scale is crucial for overall technology scaling. Shrinking transistor sizes are resulting in increased probabilities of single events causing single- and multi-bit upsets which require adoption of more complex and power hungry error detection and correction codes (ECC) in hardware. At the same time, SRAM leakage energy is increasing partly due to technology trends and partly due to the increasing number of transistors present.

This paper proposes and evaluates methods of reducing the static power requirements of caches, while also maintaining high reliability. In particular, we propose methods of applying reduced ECC techniques to data that has been identified (by programmer or compiler) as error-tolerant. This segregation, in turn, makes both the default data and the error-tolerant data more amenable to decay-based techniques for leakage control. We examine the potential of this split memory hierarchy along several dimensions. In particular, we consider the power and reliability issues inherent in the approach. Overall, we show that our approach allows the ECC requirements of future applications and caches to be met while also reducing leakage energy.

1 Introduction

Maintaining progress on semiconductor fabrication scaling has become increasingly difficult. Challenges have arisen both due to chip-wide power and thermal constraints, as well as due to difficulties in fabricating reliable transistors with predictable and dependable behaviors. Since array structures represent well over half the area and transistors on-chip, maintaining cache scaling trends is crucial for overall technology scaling.

One pressing hurdle for cache scaling is reliability. The shrinking size of transistors is resulting in increased probability of single- and multi-bit error upsets which require adoption of more complex and power-hungry error detection and correction codes in hardware.

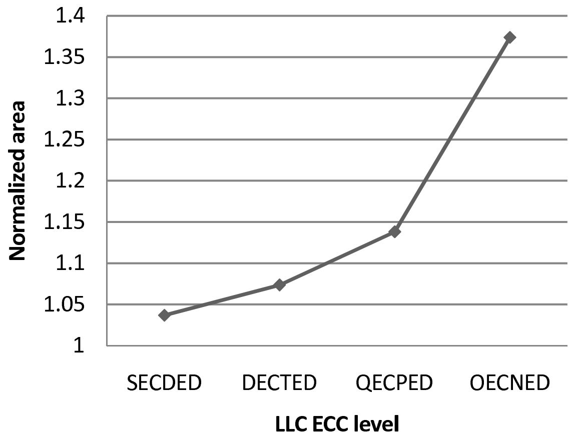
Simultaneously, another pressing hurdle for caches is the large amount of aggregate leakage power they dissipate. Schemes for improving cache reliability using error-correcting codes (ECC) exacerbate leakage problems by increasing transistor counts.

This paper explores the two challenges of cache reliability and cache leakage in a unified manner. In particular, we explore methods for implementing cache regions with either no ECC or less complex ECC in order to reduce power and area. We also explore hardware/software techniques to route data to these regions in order to maintain application performance and reliability. Finally, we also demonstrate that these caching approaches considerably improve leakage energy. Not only do they reduce transistor counts (and therefore reduce leakage energy directly) but they also have the side benefit of separating data storage in a manner that makes other leakage-control techniques like cache decay more beneficial.

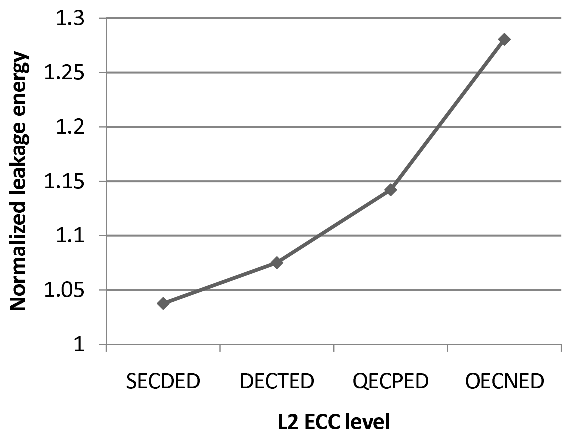
This paper explores the potential of these memory hierarchy optimizations along several dimensions. In particular, we consider the power, reliability, and usability issues inherent in the approach. Our approach offers a modest area benefit of roughly 7% for future last-level caches, while maintaining equivalent real-system reliability. Beyond this, however, its real strength emerges when paired with cache decay. By tailoring decay intervals to the distinct reference patterns for each cache partition, leakage energy can be cut in half compared to implementing cache decay on a traditional last-level cache.

Overall, we show that by viewing reliability and leakage energy as an intertwined pair of challenges, novel and highly-effective solutions can be developed to address them both.

The remainder of this paper is structured as follows. Section 2 gives an overview of the problem and our approach. Section 3 discusses our approach in more detail. Section 4 describes our experimental methodology and benchmarks. Section 5 presents our results regarding the design space for the lightweight cache, the interplay of the lightweight cache with cache decay, and compares our approach to adaptive cache decay. In Section 6 we discuss some issues that arise when using lightweight caches. Section 7 discusses related work, and we conclude in Section 8.



(a) Area Overhead



(b) Leakage Energy Overhead

Figure 1: Area (a) and leakage energy (b) for a 4MB cache with 128 byte lines and 16-way associativity as the ECC varies, normalized to values for an equivalent cache with no ECC.

2 Problem overview

2.1 Hardware Error Correction: Background and Overhead

As process technology scales down towards the nanometer domain and many more SRAM cells fit on a chip, the likelihood of reliability events in caches increases. There are several causes for these reliability issues, ranging from variations and instability in the SRAM cell itself [9] to soft errors from alpha particle or cosmic ray strikes.

ECC has emerged as a major strategy for improving cache reliability. Implementation of ECC codes in hardware enable dynamic correction of bit flips with the penalty of increased area overhead and power consumption, both of which grow as the ECC complexity increases.

Figures 1(a) and 1(b) illustrate different possible levels of ECC and the area and leakage energy overheads they incur for a 4MB level-two cache with 16-way set associativity. (We compute these using tools and methodologies described in more detail in Section 4.) Depending on the level of ECC implemented, both area and leakage energy exhibit overheads of 3-37%. While ECC offering single-error-correction and double-error detection (SECDED) is currently most common, increasing error rates are leading manufacturers to consider other more aggressive (and therefore higher overhead) approaches for the future.

2.2 Our Work

Given the significant overheads associated with ECC, it is natural to consider whether there are judicious alternatives to its widespread use throughout the cache hierarchy. In particular, our work explores a design space in which the level-2 (last-level cache or LLC) is partitioned into two regions each of which can employ different ECC approaches and different leakage control methods. For example, we explore design options in which roughly one-fourth of the LLC is implemented with only SECDED or no ECC, while the remainder of the LLC is devoted to more aggressive and more reliable ECC.

Our approach assumes that, when present in the cache at all, a memory line will be stored in either one region or another, but never both at the same time. In subsequent sections, we discuss the design issues for this approach in greater detail, including both the hardware issues of how best to build such a cache, as well as the policy issues involved in determining which data to store in each of its two regions.

2.3 Potential benefits

Before going into details about implementation issues, we first establish the potential benefits possible for such an approach. At first glance, the benefits include the area, power, and latency benefits accrued from treating the LLC as two smaller caches, and from the fact that one of these smaller caches employs little or no ECC circuitry for checking and correcting errors. Indirectly, additional benefits stem from the fact that splitting the cache into two independent regions allows each region to employ separate policies for what to store and when to evict it. Optimizing these separate policies to different data types allows our approach to offer both performance and leakage energy advantages in some cases.

For example, in Figure 2, we show the total leakage energy for combinations of traditional and lightweight caches with different ECC levels normalized to the total leakage energy of a single cache with equal data storage capacity and no error correction. In Figure 2(a), the lightweight cache does not implement any ECC while in Figure 2(b) it implements SECDED.

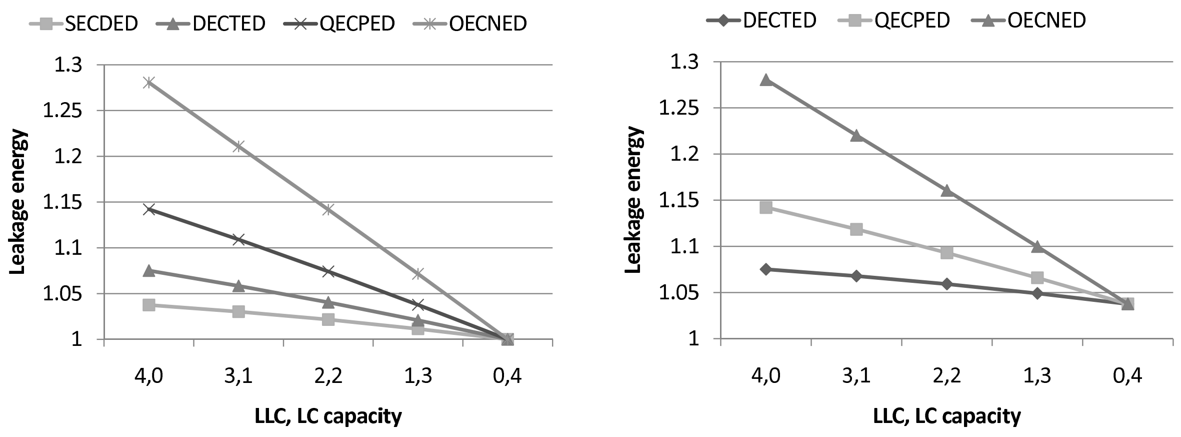
3 Design Details

Having established some of the potential benefits of our splitLLC approach, this section now treats the major design issues in more detail.

3.1 SplitLLC: Hardware Design

3.1.1 SplitLLC Size and Organization

The size and organization of the LC is obviously variable. For our examination, we choose to have two parallel caches as shown in Figure 3. In Section 5.1, we examine the benefits of using a range of cache sizes for our lightweight cache.



(a) LC: No ECC (b) LC: SECDED

Figure 2: Leakage energy for different configurations of traditional and lightweight caches, varying the ECC. The lightweight cache implements no error correction in (a) and SECDED in (b).

Our approach assumes that a memory request will be directed either to the L2 cache or the LC, but not both at the same time. We have modeled blocking caches which effectively keeps the number of ports to the next memory level constant between the two cache organizations.

Although the LC cache is likely smaller than its corresponding L2 cache and implements a lower level of ECC, we model its access latency to be the same as the L2 cache.

3.1.2 Routing References to the LC

Memory requests that do not hit in the level one cache are routed to either the L2 or lightweight cache. Data that has been designated as amenable to the LC will be mapped into particular address ranges. The hardware then shuttles accesses to these addresses to the LC. A variety of mechanisms could be used for this routing including using hardware registers to specify address ranges mapped to the LC, using techniques similar to way prediction [14], or doing parallel lookups in both caches.

3.1.3 Coherence Issues

Coherence will be maintained between the LC and other levels of the memory hierarchy via a standard coherence protocol. Because data is either annotated as LC or not, it cannot live in both the L2 cache and LC simultaneously. As a result, there is no consistency issue to maintain between these two caches.

3.2 Hardware/Software Interface Issues

A significant design issue for our splitLLC approach is that we want to be able to separate the data that is best suited for one partition versus the other. In general, our strategy follows from the following observation. While some types of data require complete correctness, many applications, especially those with streaming data accesses, contain data that can either tolerate occasional errors or that can have correct versions easily retrieved from lower levels of the memory hierarchy.

Error-tolerant data comes in several forms. In some cases, high-level application characteristics make them resilient to modest data error rates. These include image, video, and voice processing applications

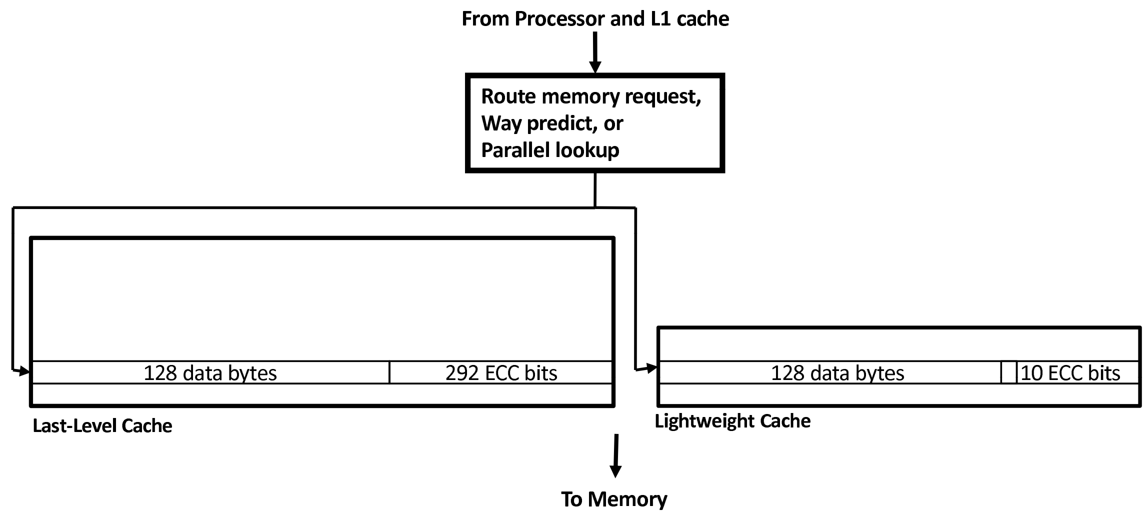


Figure 3: Both the LLC and LC have 128 byte cache lines. The LLC implements an OECNED (8-bit error correction, 9-bit error detection) ECC while the LC implements a SECDED (single-bit error correction, double-bit error detection) ECC.

where a single error in the data stream is unlikely to be detectable by the application's users. In other cases, data errors are masked by the fact that they occur in data after its last read (i.e. dead values).

Our work exploits this error tolerance by directing error tolerant data to a cache that implements no or lower levels of error correction than the primary cache. In this paper, we evaluate the potential of such hierarchies based on a suite of hand-annotated applications, but Section 6 also explores automatic, transparent techniques for either the compiler or hardware to identify such data.

4 Methodology

4.1 Area and Power Estimates

We use CACTI 6.0 [12] to estimate area and leakage energy for different cache sizes and ECC levels. To estimate the impact of ECC on leakage energy, we increase the capacity of a cache and the cache line size to accomodate the necessary ECC bits. The quantity of error correction bits needed for each ECC is based on the standard BCH codes described in [10], using 256 bit words.

4.2 Simulator

We use the Intel Pin system [11] to dynamically generate instruction traces. Pin performs run-time binary instrumentation of applications, enabling execution information to be analyzed by routines referred to as Pintools. Our Pintool models multiple levels of configurable data caches. Table 1 shows the primary configurations used in our experiments; in Section 5.1, we vary the capacity of the L2 and lightweight caches. For the purposes of this study, we are only modeling uniprocessor systems with two levels of cache, but our system can be easily extended to study multiprocessor systems and/or more levels of cache. Our experiments are run on systems with Intel Xeon processors, running Red Hat Enterprise Linux 4.6.

Instruction cache	Not modeled
L1 data cache	32KB, 64B line size, 8-way set associative, LRU replacement, 1 cycle latency
L2 data cache	3/4MB, 128B line size, 16-way set associative, LRU replacement, 10 cycle latency
Lightweight cache	1MB, 128B line size, 16-way set associative, LRU replacement, 10 cycle latency
Memory	100 cycle latency

Table 1: Memory system parameters

Our Pintool counts all instructions and forwards memory requests to the routines modeling our memory system. Although Pin does not instrument the operating system, it provides information about system calls and their associated parameters; we use this information about read and write system calls to keep track of the infrequent operating system calls accessing application data.

Rather than mapping data amenable to the lightweight cache into specific memory address ranges, the Pintool identifies this data via instruction addresses used to allocate and/or initialize this data and then subsequently directs accesses to this data to the LC. Accesses to all other data are handled by the L2 cache.

4.3 Benchmarks

We evaluate our approach using four applications, three from the MineBench [13] data mining benchmark suite and one from the PARSEC [3] benchmark suite. The three data mining applications, *Apriori*, *SVM-RFE*, and *Utility-mine*, and the H.264/AVC video encoder application *x264* are all computation and memory intensive. According to [15], the three data mining applications exhibit data access patterns that resemble streaming in which read accesses dominate, making these applications good candidates for our technique. x264 clearly deals with video data, making it a candidate for reduced ECC on its video data. Table 2 describes general application statistics regarding instruction count, memory access frequency, and memory footprint size as well as the quantity of the data mapped to the LC.

Data considered to be amenable to the LC was determined by examining each application for streaming, predominantly read only data accesses and/or data that can tolerate some level of noise. Structures that store data read in from input files are likely candidates. After initialization, these data structures become predominantly read-only; the applications create other data structures to store results during processing. As an example, for the SVM-RFE benchmark, we annotated the `svm_problem` structure which contains a two dimensional array of `svm_nodes`, each containing values read in from the input file. For the x264 application, we chose the buffers into which the input picture data is stored as well as the frames where image data remains buffered during processing.

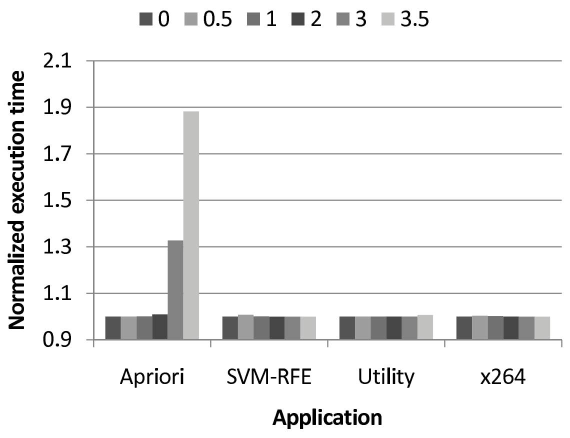
	Apriori	SVM-RFE	x264	Utility
Instructions (Bil.)	1.886	75.864	10.147	19.368
Memory Footprint(MB)	100.4	45.1	15.0	503.0
Accesses (Bil.)	0.68	17.54	1.58	4.29
Amenable to Lightweight Cache				
Memory Footprint(MB)	53.7	43.9	13.3	496.4
% of Accesses	34	94	39	33

Table 2: Application data characteristics

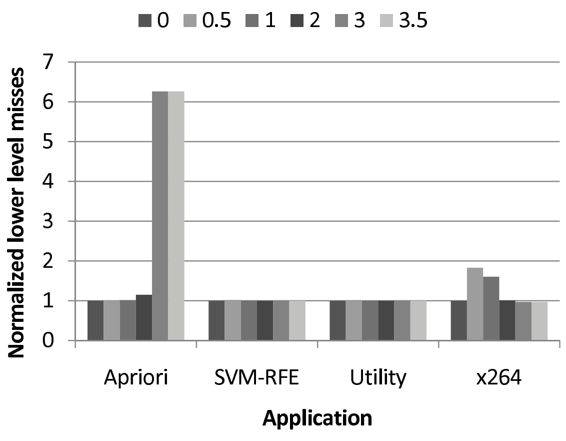
5 Results

5.1 Basic design issues

Our first objective is to determine an appropriate LC size. Because directing large numbers of data accesses to a small, parallel cache may result in changes in lower level misses which impact total system power, we evaluate the impact of using different sizes of lightweight caches by examining changes in both execution time and total number of lower level misses. Figure 4 shows these results for configurations in which the L2 and lightweight caches have a combined capacity of 4MB of data storage.



(a) Execution Time



(b) Lower Level Misses

Figure 4: Normalized execution times (a) and lower level misses (b) as the size (in MB) of the LC is varied while keeping total combined cache capacity constant at 4MB.

Despite the large memory footprints of the data mapped to the LC, ranging between 13 and 496MB as shown in Table 2, a LC of 1MB appears sufficient for these applications. Apriori begins to suffer increased execution time and lower level misses when the capacity of the LC is 2MB or more. We conclude that Apriori needs to keep the LC capacity below 2MB in order to satisfy the capacity needs of non-LC data. x264 experiences a jump in lower level misses, although not execution time, when the LC is 0.5MB. This increase in misses quickly recedes as the LC capacity increases. Thus, x264 needs its LC to have more than 0.5MB but less than 2MB of data capacity.

Assuming a partitioning of the lower level cache capacity such that we have a 1 MB LC, we can save up to 7% of the leakage energy increase experienced when error correction codes are added to a 4MB cache with no error correction.

5.2 A look at reliability

Reducing the level of ECC in the LC brings into question the issue of reliability. How can you insure the integrity of data in the LC? In choosing data to map into the LC, we have carefully chosen data that will be more amenable to reduced ECC. For example, image processing data is naturally error-tolerant; it has to handle noise introduced into images. Consequently, applications with this type of data will be able to tolerate an accidental bit flip.

The other data we selected is predominantly read-only data. For this data, a lower level of ECC such as SECDED (1 bit error correction and two bit error detection) may be sufficient to handle errors as data will not be lost if a cache line must be retrieved from a lower level of memory due to an error. The stream-like nature of the data we have selected also makes it unlikely to be retained in the LC for long periods of time, reducing the time during which errors can accumulate.

We can further harness this use of time to reduce the accumulation of errors, and consequently improve reliability, by actively removing data from the LC through cache decay. Partitioning the cache into a LLC and a LC enables us to institute more aggressive decay strategies than could be used in a single LLC without performance penalties. The cache decay intervals being used in the LC are on a much smaller scale than expected FIT estimates.

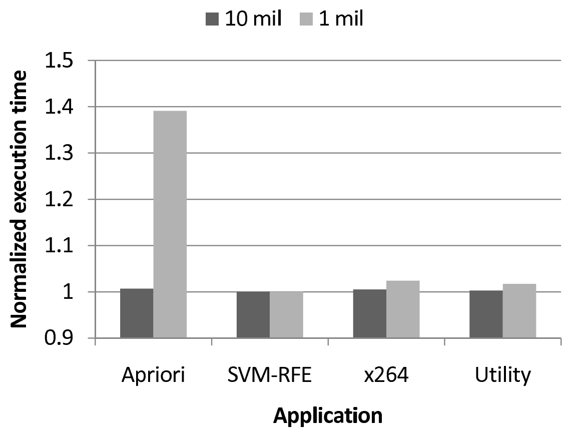
However, it is important to note that cache decay does not provide a strict guarantee on how long data will be retained in the LC depending on coherence and inclusion issues. For example, data in the LC could be actively used in a higher, inclusive cache, extending how long it remains in the LC beyond the LC's cache decay interval. To address this issue, we note that not all caches implement inclusion any more.

One alternative to cache decay that would counteract these issues would be to use a strict expiration time that invalidates data in the LC after some predefined time. However, expiring the data in the LC even though it is being actively used can have a negative performance impact if some data remains actively used longer than most data in the LC. For our four applications, we found that expiration times of 1 million cycles could increase the number of lower levels misses 1 to 45% compared to using cache decay with the same interval. Because of this sensitivity to data usage, we have chosen to explore the possibilities of cache decay instead of strict expiration. To maintain reliability in the rare cases when data persists beyond the duration of the decay interval, one could implement a very long expiration interval in the LC. This would enable you to get better performance from using cache decay than a strict expiration policy without sacrificing reliability.

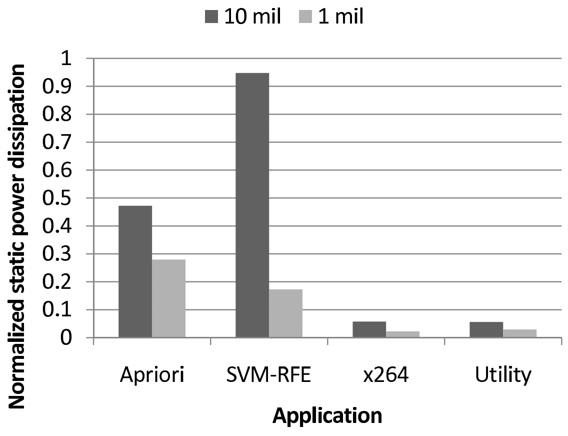
5.3 Leakage benefits of decay-based LC approaches

We now examine the impact of cache decay on these applications when only a L2 cache is used. Figure 5 shows the execution time and static power dissipation for our four applications when cache decay intervals of 10 and 1 million cycles are applied, normalized to values for a L2 cache with no decay. Figure 5(a) shows that a more aggressive decay strategy can negatively impact the execution times of three of the applications despite providing larger reductions in static power dissipation as seen in Figure 5(b). If the data mapped to the LC has characteristics consistent with these applications' overall data, it would be impossible to use a more aggressive decay interval to improve reliability in the LC and reduce static power dissipation without performance suffering.

Table 3 shows characteristics of the data considered amenable to lightweight caches and of all remaining data, including stack data. We define an active interval to be a period of time during which data is used repeatedly; active intervals are separated from other active intervals by a period of time lasting at least one



(a) Execution time



(b) Static power dissipation

Figure 5: Execution times (a) and static power dissipation (b) as the cache decay interval in the L2 cache varies, normalized to values for a L2 cache with no cache decay. The static power dissipation values assume OECNED ECC in the L2 cache.

million instructions in which the data is not accessed. If the median active interval is relatively short, it means that data is used briefly and then goes idle for at least one million cycles. The number of idle periods is the number of times a given piece of data remains idle for more than one million consecutive instructions.

The data targeted as amenable to a lightweight cache has shorter active intervals than the rest of the data in these applications according to the characteristics presented in Table 3; for SVM-RFE, we see that data targeted for the LC can be used for as little as 182 consecutive instructions before going unused for more than one million cycles. Additionally, the number of times this data is reused differs from the remaining data. For example, in Utility-mine, the median number of active intervals and idle periods are both greater for non-lightweight cache data than for the lightweight cache data. Thus, the data targeted as amenable to the LC can be viewed as having different characteristics than the other data in the applications. Consequently, it may be possible to improve reliability by aggressively decaying data and obtain larger reductions in static power dissipation without sacrificing performance by mapping this data to the LC.

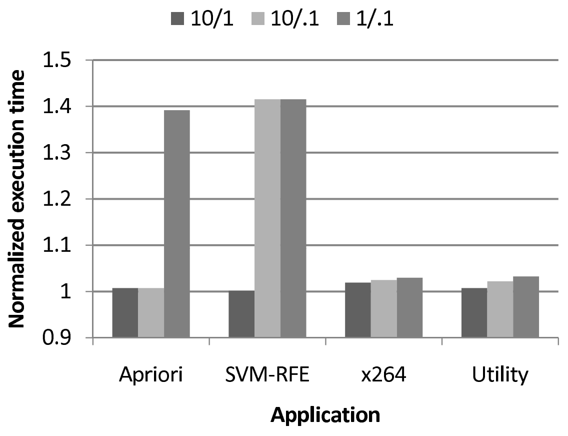
	Apriori	**SVM-RFE**	**x264**	**Utility**
Amenable to Lightweight Cache				
Median Active Interval Duration (Instr.)	1,668	182	252,755	1
Median Number of Idle Periods	2	2,146	6	1
All Remaining Data				
Median Active Interval Duration (Instr.)	412,706	340,250	811,540	362,215
Median Number of Idle Periods	2	31	4	2

Table 3: Usage characteristics of LC vs. non-LC data

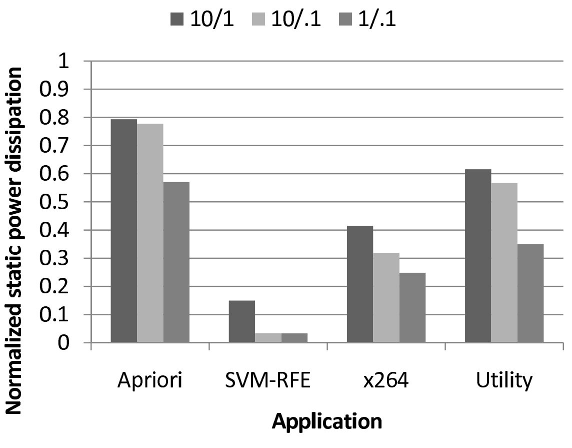
Figure 6 shows the execution times and static power dissipation for these applications when we apply more aggressive decay intervals in the LC than in the L2 cache. The decay intervals are annotated as L2 decay interval / LC decay interval; 10/1 means a decay interval of 10 million cycles was used in the L2 cache and a decay interval of 1 million cycles was used in the lightweight cache. Execution times are normalized to the use of only a L2 cache with no decay while static power dissipation is normalized to the values for a L2 cache using a 10 million cycle decay interval.

The execution times of all of the applications are only slightly worse (0-1%) than the execution times when a decay interval of 10 million cycles was used solely in the L2 cache and better than when the more aggressive 1 million cycle decay interval was used in the L2 cache. A major contributor to the difference in execution time between our multi-decay approach and using a single more aggressive decay approach is the reduction in lower level misses. For example, the number of lower level misses increases in Apriori only 11% when our mixed decay approach is used compared to a more than 700% increase when a one million cycle decay interval is applied to the entire cache. x264 and Utility-mine experience smaller but significant reductions as well.

At the same time, Figure 6(b) shows that the use of the more aggressive decay interval in the LC allows us to obtain larger static power dissipation reductions than was possible with the less aggressive 10 million cycle decay interval. Using a combination of 10/1 enables a reduction of approximately 20% of the remaining static power dissipation for Apriori, 85% for SVM-RFE, 60% for x264, and 40% for Utility. These results also suggest that our goal of aggressively decaying data in order to prevent the accumulation of errors in the LC is feasible.



(a) Execution time



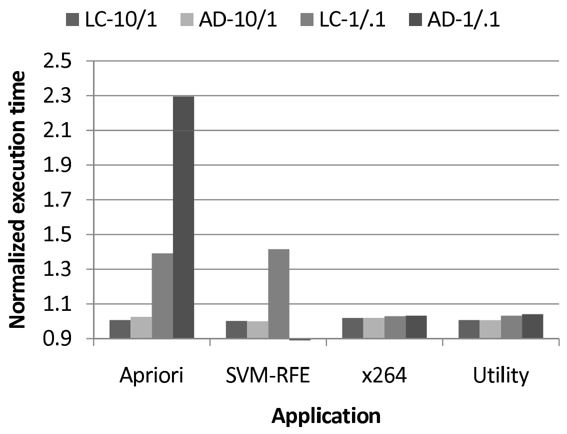
(b) Static power dissipation

Figure 6: Normalized execution times (a) and static power dissipation (b) as the cache decay interval in the L2 and lightweight caches vary. Execution times are normalized to using a L2 cache with no decay. Static power dissipation is normalized to use of only a L2 cache with a 10 million cycle decay interval.

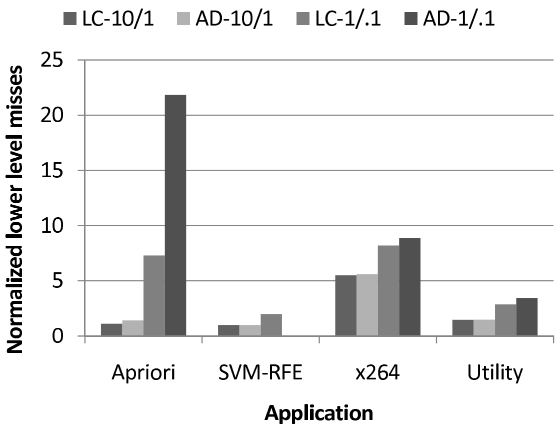
5.4 Comparison to adaptive cache decay

Given that the use of the lightweight cache enables us to tailor our cache decay strategy to specific types of data, we also compare the use of a lightweight cache to the effects of using adaptive cache decay [5]. The adaptive cache decay approach proposed in [5] provides a set of decay intervals; a cache line's given decay interval changes based on whether or not decayed cache lines stay powered off (meaning the decay was considered successful) for some fraction of the current cache decay interval.

Figure 7 shows execution times and lower level misses for these applications when adaptive cache decay (AD) is introduced to the traditional L2 cache. Overall, we see that the use of adaptive cache decay can



(a) Execution Time



(b) Lower Level Misses

Figure 7: Execution times (a) and lower level misses (b) for different cache configurations with different adaptive cache decay techniques.

increase both the execution times and lower level misses compared to using a lightweight cache. These differences can be small, as is the case when the the decay intervals are 10 and 1 million in the L2 and LC respectively, but they can become more substantial when the decay intervals are too aggressive.

Figure 8 shows the change in static power dissipation assuming the L2 cache implements OECNED ECC and the LC implements SECDED ECC. For x264 and Utility, the differences in power dissipation are small. Adaptive cache decay is able to achieve better reductions in static power dissipation for Apriori at the cost of slightly increased execution time and lower level misses.

Thus, our LC approach has the ability to achieve similar static power dissipation reductions as adaptive cache decay. However, adaptive cache decay has the potential of creating much larger performance penalties than we have observed with the LC approach when the decay intervals are chosen to be too aggressive.

Implementing adaptive cache decay in a multi-way associative cache may also have some challenges. In [5], adaptive cache decay was implemented in a direct-mapped cache. In order to determine whether or not the decay of a cache line was considered a mistake or not, the authors needed to keep tags active for decayed cache lines. Rather than keeping tags active, the work chose to create a heuristic in which the turning on of a decayed cache line before the appropriate timeout signified a mistake regardless of whether or not the data being brought into the decayed line was the same data that was decayed in the line. In a cache with set-associativity, it becomes more difficult to correlate decay mistakes with specific cache lines in order to update the decay interval for each line without maintaining tags. The need to maintain tags, however, would reduce the power savings. This complexity can be avoided by using our LC approach.

6 Discussion

6.1 LC Capacity Issues

One current limitation of our approach is the finite capacity of the lightweight cache. When the LC working set size is larger than the LC's capacity, applications will suffer capacity misses even though there may be space available in the main LLC. This potential performance penalty may inhibit software's desire to annotate data as appropriate for the LC.

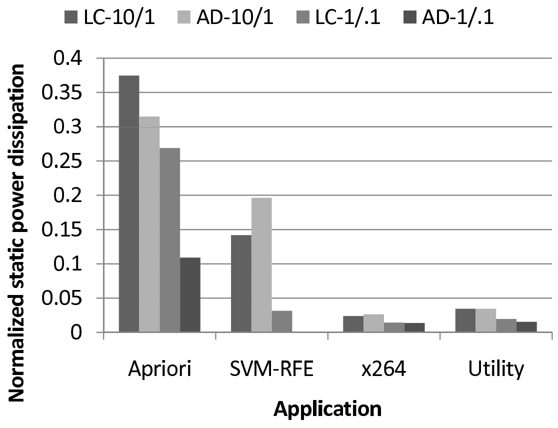


Figure 8: Comparison of LC and adaptive cache decay assuming a L2 cache that implements OECNED ECC and a 1MB LC implementing SECDED ECC.

Future implementations of our proposal can be more adaptive in determining where to place data annotated as being appropriate for the LC. For example, LC data should be placed in the LLC if it can be deduced that LC demand exceeds capacity but there is capacity available in the LLC. One possibility is to discern such situations using cache decay. For example, from the cache decay counters, it may be possible to detect the LC has reached capacity (no decayed lines) and the LLC has available capacity (multiple decayed lines) to hold overflow LC data. To enable this functionality, the mechanism for routing accesses to either the LC or LLC must support lookups in both portions of splitLLC.

6.2 Automating data annotation

Our work has shown the potential of SplitLLC approaches for a suite of applications in which the error-tolerant data has been hand-annotated. This subsection sketches out possible approaches for automating these annotations.

We have used several characteristics for determining which data to map to the LC via hand annotations. First, we considered data accessed in a streaming fashion. This implies there is a large set of data that is accessed for brief periods of time. We also required data to be error tolerant, like image and audio data, or to be predominantly read-only.

While it may be difficult to assume the data is resistant to the introduction of noise via bit flips, a single flag could be added to indicate this situation to the compiler. The compiler could then identify large contiguous allocations of array like structures accessed in a semi-regular (loop-based) fashion. In cases where data was not error tolerant, the compiler would need to determine whether this data was predominantly read-only by examining the types of accesses made to this data in conjunction with a call graph. Data written early in the call graph (during initialization) but then later accessed only via reads could be considered candidates for the LC.

A prediction scheme could be created in hardware which determines contiguous address ranges of data to map to the LC. Relatively large address ranges containing cache lines that were rarely written and were quickly decayed could be used to indicate data that could be sent to the lightweight cache. Memory addresses contiguous to existing ranges could be predicted as LC candidates. Subsequent writes to this data could result in reevaluation of these predictions.

7 Related Work

Because this work jointly considers issues from both reliabilty and power-efficiency areas, we discuss here some of the most pertinent work from each domain.

Considerable work recently has focused on more effective reliability for memory hierarchies. Some work has focused on particular error types and microarchitectural or cache management approaches for reducing their prevalence [1, 16]. In some cases, research has focused on methods particularly aimed at soft error reduction [17]. Other work has focused on aggressive methods for scaling SRAM cells and for using these scaled designs [9]. Such work, however, typically does not simultaneously consider ECC issues.

Another body of recent work has re-examined ECC methods in the face of their increased on-chip use for improving error tolerance in future generation caches [6, 7]. This work primarily looks at making ECC techniques more effective against multiple errors that are bursty in space or time. Our work is largely orthogonal to such efforts.

Our work particularly aims to balance the three concerns of performance, reliability, and leakage power. Some prior work has specifically focused on joint optimization of performance and reliability [2]. Other prior work has, like ours, jointly considered reliability and leakage concerns [8, 4]. We know of no other work, however, that considers the impact of splitting the LLC into more- and less-reliable components, both for reliability and leakage management.

8 Conclusions

This paper explores ways to reduce leakage energy associated with error correction codes implemented in lower level caches (LLC) while maintaining reliability. In particular, we introduce the concept of a SplitLLC where a region of the lower-level cache, called the lightweight cache, implements a less complex error correction code, reducing leakage energy.

We map application data that is error tolerant to this reduced ECC cache area. Examples of data amenable to this lightweight cache are streaming data which naturally includes noise, such as image and audio data, and which are predominantly read-only, such as file input data used in data mining applications. Despite applications mapping tens to hundreds of MBs of data to this reserved area, we find that 1MB is typically a sufficient capacity for this lightweight cache.

In addition to the basic SplitLLC proposal, our paper also explores the addition of cache decay techniques to both the lightweight cache and its full-ECC counterpart. First, we note that cache decay can intelligently limit data lifetimes in the lightweight cache, which increases the likelihood of its removal before it accumulates multiple errors. Furthermore, we also find that cache decay is effective in both regions of the SplitLLC, because the segregation of data between the two cache structures enables designers to tailor cache decay for each type of data. Specifically, we find that more aggressive decay intervals can be applied to the streaming data in the lightweight cache. This approach has the added benefit of improving the overall reductions in static power dissipation gained via cache decay. Overall, these techniques cut leakage energy in half compared to implementing cache decay on a traditional last-level cache. Our SplitLLC approach represents one step towards leakage and reliability management for large last-level caches, and is particularly promising for the growing number of image and video processing and data mining applications that access data in a streaming fashion.

References

[1] J. Abella, X. Vera, O. Unsal, and A. Gonzalez. Nbti-resilient memory cells with nand gates for highly-ported structures. Presentation slides from WDSN 2007 Workshop on Dependable and Secure Nanocomputing (in conjunction with DSN'07).

[2] G.-H. Asadi, V. S. Mehdi, B. Tahoori, and D. Kaeli. Balancing performance and reliability in the memory hierarchy. In *Proc. of Intl. Symp. on Performance Analysis of Systems and Software*, pages 269–279, 2005.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *Princeton University Dept. of Computer Science Technical Report*, 2008.

[4] V. Degalahal, Lin Li, V. Narayanan, M. Kandemir, and M.J. Irwin. Soft errors issues in low-power caches. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, pages 1157–1166, October 2005.

[5] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. In *Proc. of Intl. Symp. on Computer Architecture*, pages 240–251, 2001.

[6] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe. Multi-bit error tolerant caches using two-dimensional error coding. In *Proc. of Intl. Symp. on Microarchitecture*, pages 197–209, 2007.

[7] S. Kim and A. K. Somani. Area efficient architectures for information integrity in cache memories. In *Proc. of Intl. Symp. on Computer Architecture*, pages 246–255, 1999.

[8] L. Li, V. S. Degalahal, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Soft error and energy consumption interactions: A data cache perspective. In *Proc. of Intl. Symp. on Low Power Electronics and Design*, August 2004.

[9] X. Liang, R. Canal, G. Wei, and D. Brooks. Process variation tolerant 3t1d-based cache architectures. In *Proc. of Intl. Symp. on Microarchitecture*, pages 15–26, 2007.

[10] S. Lin and D. J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice Hall, 1983.

[11] C. Luk, R. S. Cohn, R. Muth, H. Patil, A. Klauser, P. G. Lowney, S. Wallace, V. J. Reddi, and K. M. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *Proc. of Conf. on Programming Language Design and Implementation*, pages 190–200, 2005.

[12] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *Proc. of Annual Intl. Symp. on Microarchitecture*, pages 3–14, 2007.

[13] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. N. Choudhary. Minebench: A benchmark suite for data mining workloads. In *IISWC*, pages 182–188, 2006.

[14] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, and K. Roy. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In *Proc. of Intl. Symp. on Microarchitecture*, pages 54–65, 2001.

[15] K. A. Shaw. Understanding the working sets of data mining applications. In *Eleventh Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-11)*, 2008.

[16] J. Shin, V. Zyuban, P. Bose, and T. M. Pinkston. A proactive wearout recovery approach for exploiting microarchitectural redundancy to extend cache sram lifetime. In *Proc. of Intl. Symp. on Computer Architecture*, pages 353–362, 2008.

[17] V. Sridharan, H. Asadi, M. B. Tahoori, and D. Kaeli. Reducing data cache susceptibility to soft errors. *IEEE TRans. on Dependable and Secure Computing*, 3:353–364, Oct 2006.

[18] X. Vera, J. Abella, A. Gonzalez, and R. Ronen. Reducing soft error vulnerability of data caches. In *SELSE 2007 3rd Workshop on Silicon Errors in Logic - System Effects*, 2007.