Energy-Efficient Cache Design Using Variable-Strength Error-Correcting Codes

Alaa R. Alameldeen, Ilya Wagner, Zeshan Chishti, Wei Wu, Chris Wilkerson, Shih-Lien Lu

Intel Corporation
2111 NE 25th Ave M/S JF2-04, Hillsboro, OR 97124, USA

{alaa.r.alameldeen, ilya.wagner, zeshan.a.chishti, wei.a.wu, chris.wilkerson, shih-lien.l.lu}@intel.com

ABSTRACT

Voltage scaling is one of the most effective mechanisms to improve microprocessors' energy efficiency. However, processors cannot operate reliably below a minimum voltage, Vccmin, since hardware structures may fail. Cell failures in large memory arrays (e.g., caches) typically determine Vccmin for the whole processor. We observe that most cache lines exhibit zero or one failures at low voltages. However, a few lines, especially in large caches, exhibit multi-bit failures and increase Vccmin. Previous solutions either significantly reduce cache capacity to enable uniform error correction across all lines, or significantly increase latency and bandwidth overheads when amortizing the cost of error-correcting codes (ECC) over large lines.

In this paper, we propose a novel cache architecture that uses variable-strength error-correcting codes (VS-ECC). In the common case, lines with zero or one failures use a simple and fast ECC. A small number of lines with multi-bit failures use a strong multi-bit ECC that requires some additional area and latency. We present a novel dynamic cache characterization mechanism to determine which lines will exhibit multi-bit failures. In particular, we use multi-bit correction to protect a fraction of the cache after switching to low voltage, while dynamically testing the remaining lines for multi-bit failures. Compared to prior multi-bit-correcting proposals, VS-ECC significantly reduces power and energy, avoids significant reductions in cache capacity, incurs little area overhead, and avoids large increases in latency and bandwidth.

Categories and Subject Descriptors

B.3.4 [**Memory Structures**]: Reliability, Testing, and Fault-Tolerance. B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance.

General Terms: Performance, Design, Reliability.

Keywords: Cache design, error-correcting codes, variable-strength codes, low-voltage design.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ISCA'11, June 4–8, 2011, San Jose, California, USA.
Copyright 2011 ACM 978-1-4503-0472-6/11/06...$10.00.

1. INTRODUCTION

Energy efficiency is becoming the key design concern for modern computer systems. Modern processors use low-power operating modes that optimize power in addition to the normal, high-performance operating mode. Reducing supply voltage is one of the most effective methods to lower power consumption. However, as supply voltage decreases, the effects of process variations become more pronounced, causing many circuits to fail. Such variations restrict voltage scaling to a minimum value, Vccmin, which is the minimum supply voltage for a die to operate reliably [27]. Failures in large memory structures (e.g., caches) typically determine Vccmin for the whole processor [27].

Prior work has explored techniques to enable ultra-low voltage cache operation in the context of high memory cell failure rates [1][2][9][22][27]. At such voltages, there is a high probability that one or more cache lines will exhibit multi-bit failures. Consequently, these techniques sacrifice a significant percentage of cache area (up to 50%) to enable correcting failures at low voltages using redundancy or error-correcting codes (ECC). This high area overhead is necessary to account for the uniform capability to correct multi-bit failures in any cache line. A recently proposed solution, Hi-ECC [26], amortized the cost of multi-bit error-correcting codes on larger (1KB) cache lines to reduce the cache area overhead to less than 2% (in the context of reducing refresh rates of large eDRAM caches). However, Hi-ECC suffers from higher latency and bandwidth consumption due to reading larger cache lines, and the need to perform a costly read-modify-write operation on every write to update ECC bits, thereby making it less appealing for small cache structures.

In this paper, we make the observation that the probability of a cache line having a multi-bit error is significantly lower than the probability of having zero or one failures. Consequently, selective high-strength protection of a few cache lines provides a significantly better usage of the ECC bit budget than fixed-strength, uniform techniques. To enable an area-efficient cache design that operates with no additional latency or bandwidth at low voltages, we propose using multi-bit ECC only on cache lines that need such protection. Our solution, variable-strength ECC (VS-ECC), allocates single-error correcting, double-error detecting (SECDED) ECC to all cache lines, while reserving a few more bits per cache set to one or more lines that can exhibit multi-bit failures. The minimum SECDED protection allocated to all lines ensures that our design can recover from all single-bit non-persistent failures (e.g., due to soft errors or erratic failures [9]). Since SECDED is a standard feature in many existing cache designs [5][23], the area overhead of VS-ECC is limited to the additional bits and logic needed to store and process the strong multi-bit ECC. By providing strong protection only where necessary, VS-ECC can reduce operating voltage compared to

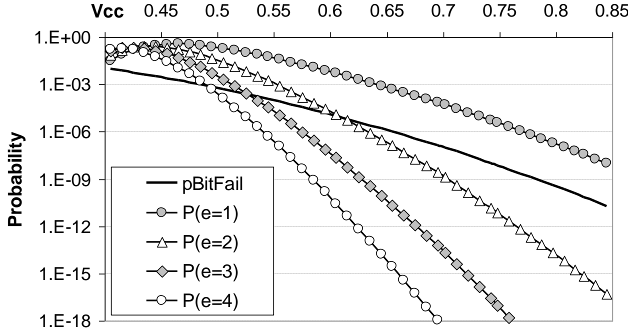

Figure 1. Probability of a single bit failure and probability P_e of e failures in a 64B cache line.

fixed-strength ECC at the same area budget, resulting in lower power consumption. Furthermore, since the fraction of lines using the high-latency multi-bit ECC is small, the performance penalty of VS-ECC is insignificant.

The main challenge for VS-ECC is to identify which cache lines exhibit multi-bit failures at low voltage and need the extra ECC protection. We propose a mechanism that uses multi-bit ECC to protect a portion of the cache after the first transition to the low-voltage mode, while comprehensively exercising the remaining lines with a standard memory testing algorithm. As a result, our hybrid characterization solution addresses two main constraints of online testing: the need to extensively stress every bit (and combinations of different bits) with pre-determined patterns, and the need to enable a part of the cache for fast transition to the low-voltage mode with little impact on user experience.

In this paper, we make the following main contributions:

1. We make the observation that the vast majority of cache lines, more than 96%, only exhibit zero or one failures at voltages as low as 500 mV.
2. We propose a novel cache architecture that uses variable-strength ECC (VS-ECC). The cache uses a simple ECC mechanism for the common case, while allocating a few bits per cache set to enable multi-bit corrections for the lines that exhibit multi-bit failures at low voltages. We propose three alternative designs for VS-ECC with varying degrees of hardware complexity, latency, and Vccmin reduction.
3. We propose a dynamic mechanism that characterizes a cache to identify multi-bit failing lines at low voltages with little performance overhead.
4. We show that, when combined with selective line disabling, VS-ECC can achieve reliable cache operation at 500 mV for a 2 MB cache, cutting energy consumption in half compared to SECDED ECC with little impact on performance or die area. VS-ECC requires no additional latency and bandwidth to access larger cache lines, and avoids the overhead of read-modify-write operations on every cache write.

In the remainder of this paper, we discuss the impact of bit failures on Vccmin and summarize the previously proposed schemes for achieving reliability at lower supply voltages in Section 2. We explain our proposed techniques in detail in Section 3, and our error-correction strategy in Section 4. We introduce the experimental methodology in Section 5, evaluate our design in Section 6, and conclude the paper in Section 7.

2. BACKGROUND

2.1 VS-ECC Overview

Modern processors use a large portion of their die for SRAM caches, which must operate free of errors for correct software execution. However, due to variations in the manufacturing process, some of the cache bits are weaker than others and may fail when supply voltage is reduced to conserve power [16][27]. Consequently, a cache with no error mitigation capabilities cannot operate reliably below a minimum voltage, Vccmin, below which at least one cache bit fails. Simple error-correcting codes (e.g., SECDED) add ECC check bits for error tolerance, thus allowing the cache to operate at a lower supply voltage while incurring a small area overhead. However, significant voltage reductions may introduce more errors than could be handled by SECDED.

In Figure 1, we show the bit failure data from Kulkarni, et al. [16], indicating the probability of a single SRAM bit failure (pBitFail) across a range of voltages. We also plot probabilities P_e of having $e=1,2,3,4$ errors in a 64-byte cache line. By examining Figure 1, we observe that the probability of two or more bit failures in a line is very low, even at voltages as low as 500 mV. Therefore, only a small fraction of lines would need multi-bit ECC, while SECDED protection is sufficient for the remainder of the cache. This motivates the need for a variable-strength technique, VS-ECC, where the check bit budget is allocated judiciously to only a few lines that require multi-bit protection.

We further demonstrate the benefits of VS-ECC with the following example. Given a single set of a 16-way set-associative cache, we can either use a fixed-strength double-error correcting, triple-error detecting (DECTED) ECC on each line of the set, or a variable-strength ECC, where a few lines are protected stronger than the rest of the set. For this example, we assume that twelve of sixteen lines in the set are protected by SECDED ECC, while the other four lines can correct up to 4 errors (12-SECDED+4-4EC5ED). We also assume that SRAM failures are random in nature [19]. Representing the probability of a line having e errors as P_e, we compute the probability that a set works correctly with DECTED (P_{DECTED}) and the probability that a set works correctly with VS-ECC ($P_{VS\text{-}ECC}$) as follows:

$$P_{DECTED} = (\sum_{i=0}^{2} P_{e=i})^{16}$$

$$P_{VS-ECC} = \sum_{l=0}^{4} \binom{16}{l} (\sum_{i=2}^{4} P_{e=i})^{l} (\sum_{j=0}^{1} P_{e=j})^{16-l}$$

In the second equation, $l=0$ computes the probability of all sixteen lines having 0 or 1 errors:

$$P(all\ lines\ have\ 0\ or\ 1\ errors) = \binom{16}{0} (\sum_{i=2}^{4} P_{e=i})^{0} (\sum_{j=0}^{1} P_{e=j})^{16-0} = (\sum_{j=0}^{1} P_{e=j})^{16}$$

On the other hand, for $l=1,2,3,4$ we calculate the probability of having l lines with 2 to 4 errors each and the remaining lines having 0 or 1 errors. The probability of a set failure is then calculated trivially as:

$$P_{DECTED_fail} = 1 - P_{DECTED} \quad \text{and} \quad P_{VS-ECC_fail} = 1 - P_{VS-ECC}$$

Sweeping a range of voltages and bit failure probabilities, we observe that P_{DECTED_fail} is always higher than $P_{VS\text{-}ECC_fail}$ for voltages above 450 mV (Figure 2). While we used a single set in this

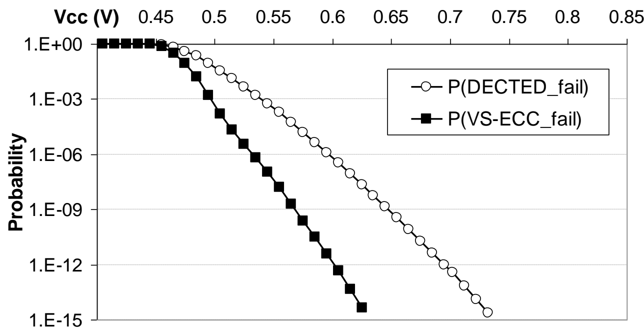

Figure 2. Probability of a single set persistent failure in a 16-way cache with DECTED or VS-ECC.

example for simplicity, this result can be extrapolated for the whole cache (Section 6). Thus, a VS-ECC-protected cache is able to operate reliably at a voltage lower than the failure point of a DECTED cache. For an entire cache with thousands of sets, the gap between the two techniques widens super-linearly. In Section 4, we demonstrate that the 12-SECDED+4-4EC5ED technique requires fewer check bits than a full DECTED ECC solution, further motivating the choice of VS-ECC. It is also worth noting that this analysis does not consider error clustering, which would benefit VS-ECC much more than a fixed-strength ECC design.

2.2 Related Work

Traditional approaches to tolerate bit defects in caches have used row/column level redundancy by adding spare rows/columns to the cache [24]. These approaches remap a row/column with one or more bit defects to a spare row/column. While this coarse-grain remapping is effective in dealing with relatively low error rates encountered during high-voltage mode, it cannot mitigate defects in thousands of cache lines during low-voltage operation.

Wilkerson, et al. [27], and Roberts, et al. [22], proposed techniques to enable reliable low-voltage operation by sacrificing a portion of the cache to save the locations of defects and the values of correct data. ZerehCache employs fine granularity re-mapping of faulty bits to tolerate high failure rates, and relies on additional manufacturing-time testing to identify the locations of defects on sub-cache line granularities before solving a graph coloring problem to find the best re-mapping of defects to spare lines [2]. These techniques incur performance overheads at low voltages due to the smaller cache capacity, higher latency, and the need to access a fault map in parallel. They also rely on manufacturing-time testing to pinpoint the location of bit defects, which would not help with soft errors.

Circuit-level approaches have been proposed to improve bit cell reliability by either upsizing transistors or using alternative SRAM cell designs, such as eight-transistor (8T) or ten-transistor (10T) cells [16]. These approaches incur significant area overhead and higher leakage during both the high-voltage and low-voltage modes. In particular, the work in [27] shows that a 10T cell would achieve a comparable operating voltage to our VS-ECC but with a much larger area overhead (100% overhead for the 10T cell vs. less than 4% overhead for VS-ECC).

Many previous papers have proposed the use of error-correcting codes (ECC) to improve cache reliability [9][14][26]. Kim, et al. [14], proposed two-dimensional ECC to correct multi-bit errors. 2D-ECC is tailored towards clustered bit defects and is less effective in dealing with high rates of random defects. Chishti, et al. [9], proposed multiple-bit segmented ECC (MS-ECC) at fine sub-cache line granularities to tolerate high failure rates. MS-ECC sacrifices a large portion of the cache (25-50%) to store ECC check bits for the rest of the lines, which leads to performance loss during the low-voltage mode. Furthermore, MS-ECC requires 25-50% of the cache to be flushed and ECC check bits to be computed for the rest of the cache every time we switch from the high-voltage to the low-voltage operating mode. Yoon and Erez proposed reducing the cost of storing error-correcting codes for the last level cache by partitioning the code between the cache and memory [28]. Our previous work, Hi-ECC, uses multi-bit ECC in an eDRAM cache to reduce refresh power [26]. Hi-ECC reduces the storage cost of multi-bit ECC by saving check bits over large cache line granularities (1 KB), and reduces the latency cost by separating the common case of zero or one bit failures from the uncommon case of multi-bit failures. However, large cache lines (used in Hi-ECC) result in higher dynamic power. The approach presented in this paper uses variable-strength error protection, differing from all the previous ECC-based cache reliability techniques that allocate fixed ECC budgets to all cache lines.

Variable-strength protection schemes have been studied extensively in the communication theory context, since some parts of the transmitted data have much higher importance for system stability. For instance, an error in a control or in a routing field of a message header may have much worse consequences than a bit flip in the payload. Fujiwara presents a comprehensive overview of these techniques [10]. The only cache-related study was performed by Kim and Somani [15]. They provided ECC capability for some, but not all, L1 cache lines to protect against soft errors, and considered frequently accessed lines to be most vulnerable. Compared to Kim and Somani's study, our proposal targets both transient and persistent errors at low voltages, provides a minimum level of protection for all lines, and allocates more protection to cache lines with a higher number of persistent errors. To the best of our knowledge, this paper is the first to use variable-strength error correction for energy-efficient cache design.

3. VARIABLE-STRENGTH ECC

The key idea behind VS-ECC is to employ error-correcting codes of different strengths for different cache lines, based on the number of failing bits in each line in the low-voltage mode. VS-ECC uses fast SECDED ECC for the common case of lines with zero or one errors, and resorts to a slower multi-bit correction only in the rare case of two or more errors. However, VS-ECC requires cache lines to be classified based on the number of bit failures. In this section, we discuss three variations of VS-ECC that could be implemented on top of a baseline SECDED cache. In Section 3.1, we describe the cache organization. We then discuss the cache characterization and low-voltage operation in Section 3.2 and Section 3.3, respectively.

3.1 Cache Organization

We illustrate VS-ECC with the example of a 16-way set-associative cache shown in Figure 3. The tag and data arrays are logically organized similar to traditional set-associative caches. Tags in this setup occupy only a small fraction (~10%) of the cache area and are assumed to be protected by either upsized cells or SECDED as proposed in [9] and [27]. Each cache line has an associated SECDED ECC field which contains check bits (of length l_{SECDED}) to correct one and detect two errors in the line. To support multi-bit errors, we propose the following three designs:

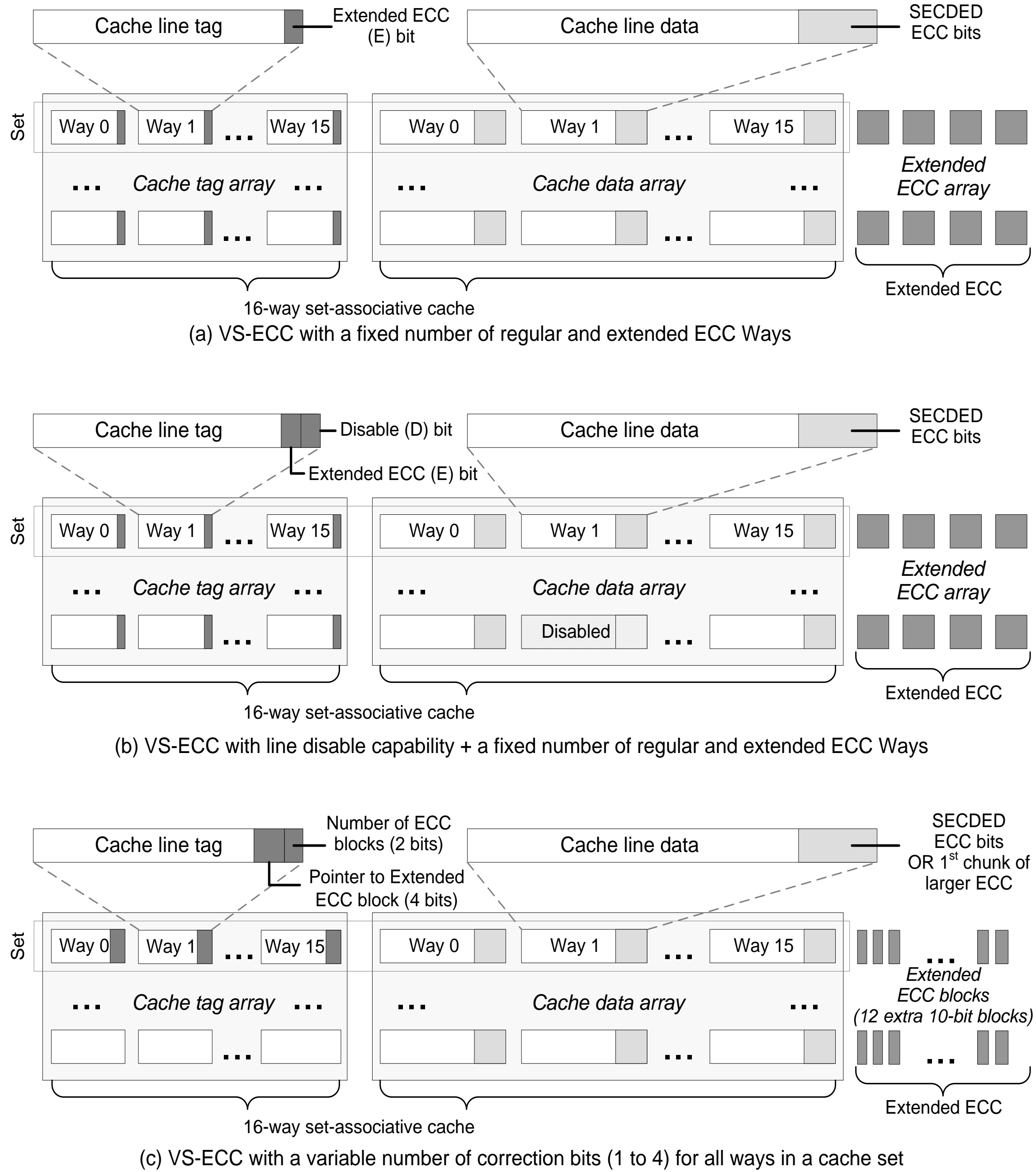

Figure 3. VS-ECC Cache organization with three alternative designs.

1. VS-ECC-Fixed augments each cache set with four extended ECC (eECC) fields, each containing ($l_{MULTI-BIT}$ - l_{SECDED}) bits, where $l_{MULTI-BIT}$ is the number of check bits required to correct a multi-bit failure in a line (Figure 3a). The four eECC fields enable the cache to use a strong ECC code for any four of the 16 ways in each cache set. To distinguish lines that use multi-bit correction from lines that use SECDED ECC, we add an extra status bit to each tag, called 'Extended ECC bit' or E-bit. If a cache line is classified as having multi-bit failures, the E-bit is set to 1, otherwise it is reset to 0. We also extend the cache with a separate multi-bit ECC encoder/decoder in addition to a SECDED encoder/decoder. We discuss the area and logic complexity of this design in Section 4.

2. VS-ECC-Disable: This design (Figure 3b) is similar to VS-ECC-Fixed with the addition of one status bit with each cache tag. This 'Disable' or D-bit is used to indicate that the corresponding cache line is not in use, or is always in the invalid state. This design provides better soft error coverage by using SECDED as a minimum level of protection for lines with no persistent failures, while using multi-bit ECC to protect lines with one or two persistent failures. For lines with more than two persistent failures, we set the D-bit and the line is no longer used in the low-voltage mode. Using at least 3-bit correction for the multi-bit ECC protects all lines from single-bit soft errors, while sacrificing a small fraction of cache capacity due to the disabled lines.

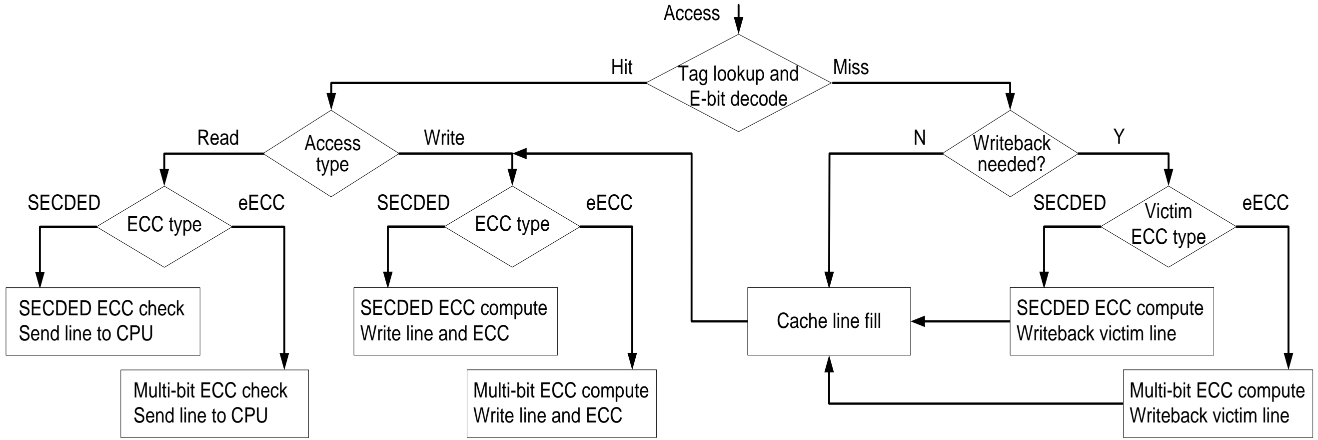

Figure 4. Cache operation in low voltage mode for VS-ECC-Fixed.

3. VS-ECC-Variable: In this design (Figure 3c), we use the SECDED bits associated with each cache line either as regular SECDED code or as the first eleven bits of a multi-bit ECC. We add two status bits per cache tag to indicate the number of ECC blocks for the corresponding cache line. This determines the line's level of protection, from 1: SECDED to 4: 4EC5ED. During cache characterization, we assign SECDED to lines with no failures, and we assign a number of ECC blocks for other lines to cover both persistent failures and soft errors.

3.2 Cache Characterization

Before VS-ECC in any of its variations can be turned on in the low-voltage mode, cache lines need to be classified on the basis of the number of bit failures. We classify cache lines by running a special runtime characterization phase during the first transition to the low-voltage mode. We chose runtime characterization over manufacturing-time testing since we want to cover a variety of persistent failures that are not present at time zero (e.g., aging faults). Furthermore, manufacturing-time characterization is expensive and requires on-die non-volatile storage to store the fault map data. The goal of the runtime testing is to extensively analyze a fraction of the cache using traditional memory characterization algorithms, while leveraging the eECC bits on the remainder of the lines to minimize the impact on system performance and user experience. In this section, we discuss characterization for the VS-ECC-Fixed configuration, but this mechanism can be trivially extended to the other VS-ECC variants. Since we have only four eECC fields per cache set, only four lines can be active and contain protected live data, while the rest of the cache is inactive and under test. Note that errors can also be discovered in the active portion, however, eECC allows us to mitigate them and to mark E-bits for active lines with multi-bit failures.

Before entering the characterization phase, we reset all the E-bits to zero, reset valid bits for inactive cache blocks, write back all the dirty data, and reduce the processor voltage to the target Vccmin. The four eECC fields then become associated sequentially with the first four ways (ways 0, 1, 2 and 3) in each set. We deactivate the remaining ways (4 through 15) by modifying the cache replacement algorithm so that it does not allocate any new blocks to them. Even though we sacrifice 75% of the capacity, we ensure that the available eECC is enough to correct any multi-bit failure in the active portion of the cache. Note that a complete cache flush on a voltage transition is not strictly required, and one can design an algorithm with lower performance impact, albeit higher complexity, where the first four ways remain in the cache and strong ECC is generated for them, while the rest of the lines are flushed.

In the active ways, on each cache write during characterization, we send the data being written to the multi-bit ECC encode logic. We split the generated multi-bit ECC between the cache line's SECDED field and its eECC field. On a read access, we send the stored data and the strong ECC to the multi-bit ECC decoder, which then outputs the corrected line data, as well as the number of encountered errors. If the number of errors is more than one, we set the line's E-bit, marking it as a line with a multi-bit failure. In this design we assume that the entire cache must operate in a single voltage domain and, consequently, high-strength protection is required for the active region. In a more sophisticated setup without this constraint, voltage can be reduced only in the portion under test, while the active portion is kept error-free at a high voltage.

In parallel, we conduct a traditional memory test on the inactive portion of the cache in the spare cycles when it is not used. Lines are written with pre-defined patterns and read back in a particular order. The testing engine, implemented in hardware as a part of the cache controller, can then identify any discrepancies between the expected and observed patterns and derive the location of failing bit(s) in each line. If a multi-bit failure is detected, we set the line's E-bit, implying that it will need extended ECC during regular operation. On the other hand, if there is only a single error, we write its location (9-bits for a 512-bit cache line) into the line's tag. In the remainder of the test, if a single-bit failure occurs again in the same line, we compare its location with the one stored in the tag using the tag matching hardware. Consequently, a 'hit' in the tag array would indicate the repetition of the same error, while a 'miss' signifies a multi-bit failure, upon which we would set the line's E-bit. The process continues until the testing algorithm completes or until we discover a set (spanning both active and inactive regions) with 5 or more E-bits set to 1 or a line with five errors. This situation indicates a case where we cannot use the entire cache in low-voltage mode because the available ECC is not enough to correct all lines with multi-bit failures, thus the cache is switched to a higher voltage and eECC fields are disabled for power savings. Alternatively, we can restart the characterization at a slightly higher voltage (e.g., 10 mV above target Vccmin). Finally, if the cache passes the test, we move live data from the active to the inactive portion and activate the latter. Testing of the previously active region is then started.

For VS-ECC-Variable, in addition to classifying the lines into single-bit and multi-bit failing lines, we need to know the exact number of erroneous bits so we can assign a sufficient number of ECC blocks. This can be done by conducting the test on one quarter of the cache at a time. Tags that belong to the portion of the cache under characterization are used to store the locations of up to three distinct errors for each of the tested lines. Note that we do not overwrite other tag status bits when we store the error locations. Thus, after a quarter of the cache is tested, the ECC block pointer bits in the tags for this portion are preserved and the test of another portion starts. While VS-ECC-Variable requires high testing accuracy to identify the exact number of persistent failures in each cache line, VS-ECC-Fixed requires a much lower accuracy since it does not differentiate between lines with 1-, 2-, and 3-bit failures. VS-ECC-Disable uses the same testing algorithm as VS-ECC-Variable since we need to know which lines have no failures, one or two failures, and more than two failures. However, this technique would still function correctly even with a lower testing accuracy.

To compute the overhead of characterization we investigated several well-known testing algorithms, including MARCH B, MARCH SS, GALPAT, and pseudo-random algorithms for testing Active Neighborhood Pattern Sensitive Faults (ANPSFs) [11][13]. In particular, we considered a deterministic (100% coverage) ANPSF algorithm and a pseudo-random coupling fault detection technique with 99.9% coverage. ANPSF ($k=5$) can detect an error caused by the interaction of up to five neighboring cells, while the coupling fault detection technique discovers erroneous interactions of any two cells in the entire cache. Based on our cache parameters (Section 5), performance data of testing algorithms, and accounting for additional testing overheads, we calculate the worst-case testing time to be below 50 seconds. This penalty, however, is incurred by our characterization only once during the first transition to the low power mode and it can be further amortized using non-volatile storage to save testing results (E-bits and eECC fields) between system reboots.

It should be noted that VS-ECC-Disable has an advantage over other VS-ECC configurations since it does not require testing with perfect coverage. It should also be noted that all VS-ECC variants are orthogonal to the exact testing algorithm used; they allow using more complex patterns at the cost of increased testing time. All VS-ECC variants mitigate the impact of runtime memory testing on system performance. Our hybrid characterization scheme ensures execution with a virtually unnoticeable penalty by keeping a portion of the cache protected and operational. The hardware overhead of characterization is trivial compared to the cache size, and involves a simple state machine with tens of state elements and a few gates.

3.3 Cache Operation

The algorithm for regular cache operation in the low-voltage mode for VS-ECC-Fixed is illustrated in Figure 4. When the cache receives a request from the processor, the cache controller first decodes the set number from the data address and reads all tags and E-bits corresponding to that set. In parallel with tag matching, we use a small decoder that determines which lines in the set use multi-bit ECC based on the values of the E-bits. Since the eECC fields associated with a set are used in the order of increasing way numbers, the decoder can quickly determine which eECC field is associated with a particular cache line by examining all the E-bits for a set. Thus, when a tag lookup completes, the cache controller knows if the line is present in the cache (hit/miss), its ECC type (SECDED or multi-bit), and which of the four eECC fields store that line's check bits.

On a read hit to a cache line that requires multi-bit ECC (E-bit = 1), the eECC bits associated with that line are sent to the multi-bit ECC decoder, in addition to the data and SECDED ECC check bits. The multi-bit decoder corrects all the errors in the line before sending it to the processor. Similarly, on a read hit to a line that requires only SECDED ECC (E-bit = 0), data and SECDED ECC check bits are sent to the SECDED decoder. On a write hit or cache fill, new ECC check bits need to be generated. If the cache line requires multi-bit ECC (E-bit = 1), then the data being written is sent to the multi-bit encoder to generate all the check bits. The first l_{SECDED} bits are written in the SECDED ECC field of the line, while the remaining check bits are written to the appropriate eECC field. Similarly, if the written cache line uses SECDED ECC, data is sent to the SECDED encoder and the generated check bits are written in the SECDED ECC field for that line.

If an access results in a cache miss and the replacement algorithm selects a dirty block for eviction, then any errors in the block need to be corrected before it is written back to the next cache level (or memory). Such evictions are conducted in a manner similar to a read request: we first check the E-bit for the victim line to decide what kind of ECC it uses. If the E-bit is set, then we send the data and the appropriate ECC bits to the multi-bit decoder. Otherwise, we send the data and the SECDED ECC bits to the SECDED decoder. After the dirty line is written back, the new line is read from the next cache level (or memory) and a fill then proceeds similar to a write request. VS-ECC-Disable and VS-ECC-Variable operate very similarly, with the difference that VS-ECC-Disable completely avoids using lines with D-bit set, and VS-ECC-Variable looks up the pointer and the size of the ECC block upon a tag match instead of using a single E-bit.

During regular operation, cache lines with zero or one failures do not incur any latency overhead, relative to a baseline cache with SECDED ECC. Lines with multi-bit errors incur an additional latency due to the higher complexity of the multi-bit encode/decode logic. However, this logic can be pipelined to speed up the operation at a higher area cost, as we describe in Section 4.

We note that VS-ECC-Disable avoids some multi-bit ECC overheads during normal cache operation by disabling lines with three or more failures. However, it still requires our dynamic characterization that relies on multi-bit error correction. Another alternative could forego hybrid characterization altogether and subject the entire cache to thorough testing, which would have an unreasonable impact on performance and user experience.

4. MULTI-BIT ERROR CORRECTION ANALYSIS

The most popular choice for ECC in memories belongs to the class of binary linear block codes [7][20], used commonly for soft-error protection. Alternatively, iterative encoding/decoding designs use much less hardware but require a higher latency [25]. Prior work has investigated fast, parallel designs for simple codes such as SECDED and DECTED [18][25]. Memory structures that can tolerate high latencies (e.g., FLASH) use stronger ECC to correct multi-bit errors [20]. As previously discussed, our solution protects the common case where cache lines have zero or one failures with simple SECDED ECC, while protecting lines with multiple failures using stronger, 4-bit-correcting, 5-bit detecting (4EC5ED) codes. In this section, we explain the ECC algorithm and analyze its complexity for both SECDED and 4EC5ED.

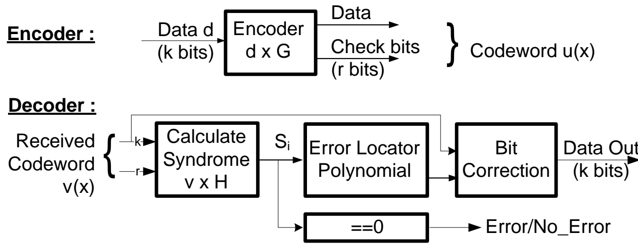

Figure 5. Overview of a BCH-based error-correcting design.

4.1 Multi-Bit BCH Codes

Binary BCH codes are a class of linear cyclic block codes that are widely used for correcting random bit-errors [20]. The simplest block code, Hamming code, is a special type of BCH code that can correct only one random error [7]. In general, a binary BCH code is defined over a finite Galois Field $GF(2^m)$, which is defined by an irreducible polynomial $p(x)$ of degree m over $GF(2)$. The field is the set of polynomials modulo $p(x)$. If α denotes the root of $p(x)$, the set of field elements can also be represented as $\{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^{\wedge}m-2}\}$. Each element α^i has a distinct representation as a polynomial with degree less than or equal to m, or simply is an m-tuple vector with m binary coefficients of the polynomial.

BCH code words are produced using a generating matrix G based on a set of generator polynomials. Checking a code word for errors involves using the H matrix (parity matrix) to obtain syndromes and check whether data has been corrupted. The detailed proof and generation of the H matrix is beyond the scope of this paper and can be found in [12][20]. Figure 5 shows a high-level block diagram for BCH error correction logic consisting of two components, the encoder and the decoder:

ECC Encoder. Input to the encoder is a k-bit data word d, represented as a vector. The encoding process is simply the multiplication of d and the generator matrix G. If the code is selected in a systematic form (i.e., part of the G matrix is the identity matrix), then data bits are not changed and will be concatenated with r check bits to obtain the final code word.

ECC Decoder. The decoder detects and corrects potential errors in the stored code word. The error-correcting logic will pinpoint the locations of all corrupted bits (if any) within the capability of the code, and then corrects them. The decoding procedure can be further divided into three steps [20]:

Step 1: Syndrome calculation. The syndrome S is the product of retrieved code word (v) and the transpose of the parity matrix H,

$$S = (Parity, S_1, S_3, \cdots, S_{2t-1}) = v \times H^T =$$

$$= v \times \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{(n-1)} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \cdots & \alpha^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \alpha^{(2t-1)} & \alpha^{2(2t-1)} & \alpha^{3(2t-1)} & \cdots & \alpha^{(n-1)(2t-1)} \end{bmatrix}$$

According to the code definition, a valid codeword v must have its resulting syndrome equal to zero. Any non-zero syndrome indicates the occurrence of one or more errors. Suppose the set of error locations is $\{i_1, i_2, \ldots, i_t\}$. More precisely, each syndrome element S_j is represented as $S_j = (\alpha^{i1})^j + (\alpha^{i2})^j + \ldots + (\alpha^{it})^j$. If the syndrome S is non-zero, we need to locate all errors in the codeword by first determining the error-locator polynomial and then solving it to pinpoint and correct errors.

Step 2: Determining the error-locator polynomial. The error-locator polynomial $\sigma(x)$ is defined such that the roots are given by the inverse of error elements $\alpha^{i1}, \alpha^{i2}, \ldots, \alpha^{it}$, respectively: $\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2 \ldots + \sigma_t x^t = (1-\alpha^{i1}x)(1-\alpha^{i2}x)\ldots(1-\alpha^{it}x)$. Berlekamp and Massey's iterative algorithm is used to obtain the σ_i from S_j by solving Newton's identities of the polynomial $\sigma(x)$ [3][17].

Step 3: Finding the error locations and correction. Solving the error-locator polynomial includes substituting the field elements α_i into $\sigma(x)$. Those substitutions that make the equation equal to zero are roots of the polynomial. Correction is done by flipping the corresponding data bits using XOR gates.

4.2 ECC Overhead Analysis

Lower latency can be achieved by parallelizing the ECC encoding and decoding algorithm. Previous studies have investigated the area and latency tradeoffs for bit-parallel binary BCH decoders [25]. Both ECC encoding and syndrome calculation (step 1 of the decoder algorithm) are based on matrix multiplication. In a completely bit-parallel implementation of the latter, each syndrome bit is obtained by a separate XOR tree with inputs taken from code words. The selection of the input bits is determined by the values in the H matrix, which is 50% full on average (half of the matrix entries are '1'). So each XOR tree's fan-in is half the number of input bits. Having a stronger correction capability does not affect the depth and latency of each tree, only increasing the number of trees.

The Berlekamp-Massey Algorithm (BMA) [6] is the best known technique to find coefficients for the error-locator polynomial (Step 2 of the decoder). BMA is based on a t-step recursive procedure, which cannot be parallelized completely. Each iteration involves a Galois field inversion that takes $2m$ steps [4]. An inversionless BMA proposed in [6] uses double the number of multiplications, but performs them in parallel and off the critical path [21].

Chien's search [8] is an elegant algorithm for Step 3 of the decoding algorithm. It leverages the cyclic nature of BCH codes, and corrects one bit per cycle with a simple linear feedback shift register. However, its latency is too high for a large data input. A faster alternative is to check all elements in parallel, which requires a large number of gates [25]. An area and latency tradeoff can be done by splitting the data bits into r chunks, so that k/r bits are tested in each cycle and the algorithm requires a total of r cycles to finish. A few additional registers, occupying negligible storage, are needed to store intermediate results.

Table 1 lists the cost for both SECDED and 4EC5ED for a 64B cache line. We implement Step 2 in a pipelined, fully-parallel and unrolled structure based on the inversionless BMA algorithm. For 4EC5ED, the algorithm takes three iterations. The first iteration takes a single cycle, and the last two iterations are split in half to fit in a clock cycle. The maximum critical path includes one GF multiplication, two GF additions and two multiplexers, for a total of five-cycle latency for this step. For Step 3, we list the overhead for two different implementations. The first approach is fully-parallelized and tests 64 bytes together, taking only one cycle. The second approach, which we use in our simulations, corrects 8 bytes per cycle. It reuses the correcting logic and reduces area overhead, but incurs a latency of eight cycles for a 64-byte line. The fully parallelized SECDED design is simple and fast, with only 14k gates of hardware and one cycle for either the encoding or the decoding process. On the other hand, the 4EC5ED design needs about 50k

Table 1. Logic and Latency Overheads.

		SEC DED	4EC5ED
Storage	(per 512-bit data)	11 bits	41 bits
Gate Count	Encode	2k XOR	10k XOR
	Decoder (Step 1)	2k XOR	11k XOR
	Decoder (Step 2)	512 XOR	3.5k XOR+4k AND+300 Flip-flops
	Decoder (Step 3)	9.2k AND	1) 123k XOR + 5.3k OR + 3.2k AND 2) 15k XOR + 640 OR + 400 AND + 640 Flip-flops
	Detection	10 XOR	40 XOR
	Total	13.7k	~160k / ~50k
Latency	Encode	8 XOR	9 XOR (1 cycle)
	Decoder (Step 1)	8 XOR	10 XOR (1 cycle)
	Decoder (Step 2)	1 XOR	65 XOR (5 cycle)
	Decoder (Step 3)	5 AND	1 cycle or 8 cycle
	Detection	4 OR	6 OR
	Decoder Total	1 cycle	7 or 14 cycles

gates and a 14-cycle latency. An additional cycle may be needed to check whether we need multi-bit correction (total of 15 cycles). Assuming a gate is equal to twice the area of a cache cell, 4EC5ED has a logic area overhead of ~0.6% for a 2MB cache.

Storage Overhead. Higher error-correcting capability requires a higher storage overhead for check bits: To correct t-bit errors and detect $(t+1)$-bit errors, a BCH code requires $t*m+1$ check bits [20]. For a certain length of data input (e.g., k-bits), the rank m is the smallest possible number that makes the total code length ($k+t*m+1$) less than the number of non-zero elements in $GF(2^m)$. For a 64B (512 bit) cache line, m is equal to 10, as $2^{10} > 512 + 10+1 > 2^9$. So a 64B (512 bit) cache line requires 11 check bits for SECDED ECC.

Consequently, in our baseline configuration with SECDED ECC on all lines to recover from soft, non-persistent errors, we have (512+11)*16 = 8368 bits per set for a 16-way cache. In our evaluation, we use VS-ECC-Fixed (12x1, 4x4) which needs twelve 11-bit SECDED codes and four 41-bit 4EC5ED codes for each set, in addition to one status bit per line. VS-ECC-Fixed therefore requires a total of 30*4+16*1 = 136 bits per set more than SECDED (1.6% overhead). This overhead is smaller than using a uniform DECTED code with 21-bits (10-bits more than SECDED) for every cache line (1.9% overhead). VS-ECC-Disable has an additional disable bit per cache line, so the total number of additional bits per set is 136+16=152 (1.8% overhead vs. SECDED). VS-ECC-Variable needs an additional six bits per cache line, for a total of 152+16*6 = 248 bits per set (3% overhead over SECDED).

Table 2. Benchmarks.

Category	Number of traces	Example benchmarks
Digital home (DH)	9	H264 decode/encode, flash
SPECINT2006 (ISPEC)	8	www.spec.org
SPECFP2006 (FSPEC)	9	www.spec.org
Games (GM)	19	Doom, quake
Multimedia (MM)	24	Photoshop, ray tracer
Office (OFF)	29	Spreadsheet/word processing
Productivity (PROD)	17	File compression, Winstone
Server (SERV)	14	SQL, TPC-C
Workstation (WS)	7	CAD, bioinformatics
Kernels (KERN)	7	Streaming, random access microbenchmarks
ALL	143	

5. SIMULATION METHODOLOGY

Simulation baseline configuration. We use a cycle-accurate, execution-driven simulator running IA32 binaries. The simulator is micro-operation (uOp) based, executes both user and kernel instructions, and models a detailed memory subsystem. As a baseline, we model an out-of-order superscalar processor similar to a single core of the Intel® Core™ i7 processor running in a single voltage domain at 2 GHz. Our memory system includes a 32 KB, 8-way set-associative L1 instruction cache, a 32KB, 8-way set-associative L1 data cache, and a 2MB, 16-way set-associative unified L2 cache. All caches in our system are configured to have 64-byte lines. Our baseline configuration uses SECDED ECC to guarantee recovery from single-bit soft errors and other non-persistent failures.

Benchmarks. We simulate ten categories of benchmarks. For each individual benchmark, we carefully select multiple sample traces that are representative of its behavior. Table 2 lists the number of traces and example benchmarks included in each category. We use instructions per cycle (IPC) as the performance metric. The IPC of each category is the geometric mean of IPCs of all traces within that group. We normalize the IPC of each category to the baseline for performance comparison, and use the average IPC and the operating voltage as inputs to a power model to estimate energy consumption.

Simulated configurations. Our baseline configuration (BASE) has SECDED ECC and a 12-cycle L2 hit latency in addition to one cycle for SECDED correction. In addition to the baseline, we model two fixed-strength ECC configurations, three variable-strength ECC configurations, and the previously proposed multi-bit segmented ECC configuration (MS-ECC) [9]. Fixed-strength configurations (DECTED, 4EC5ED) model an L2 cache augmented with ECC codes that correct two and four errors, respectively. Since we provide a recovery guarantee for all single-bit non persistent failures, DECTED can only recover from a single persistent failure per line while 4EC5ED can only recover from up to three failures per line. DECTED has the same latency as our baseline SECDED, while 4EC5ED incurs an additional latency of 15 cycles. We also simulate variations of our VS-ECC mechanism with SECDED on

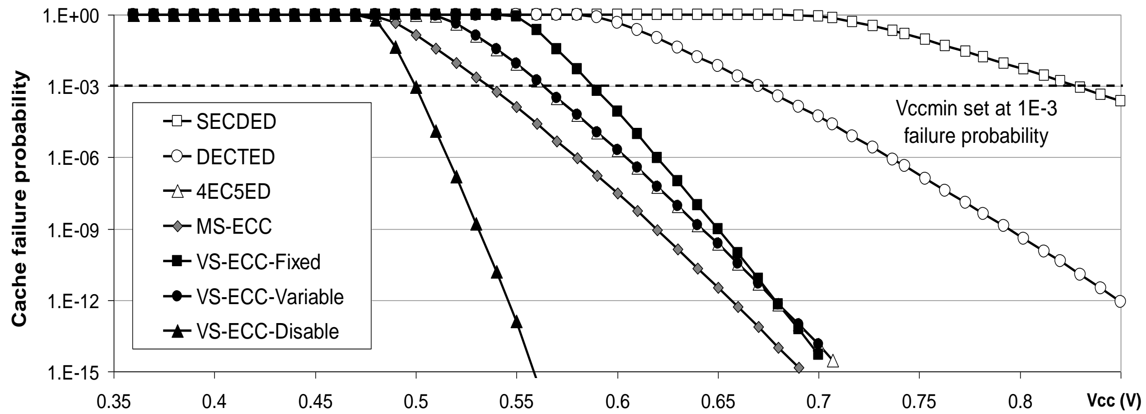

Figure 6. Failure probability for persistent failures and soft errors as a function of supply voltage for different configurations.

all lines, and multi-bit correction on a few lines per set: VS-ECC-Fixed (12x1, 4x4) has four lines per set with 4EC5ED code while the remaining lines have SECDED; VS-ECC-Variable has SECDED for all lines and 12 extra 10-bit ECC blocks; VS-ECC-Disable is the same as VS-ECC-Fixed with the ability to disable lines with multi-bit failures. All VS-ECC designs were simulated in the regular operation mode assuming multi-bit failing lines have been identified. We model MS-ECC with a 4-bit error-correcting code for each 64-bit segment of a cache line. To provide the same guarantee against non-persistent faults, each segment only protects against three persistent failures. Since MS-ECC uses the latency-efficient but area-inefficient Orthogonal Latin Square Codes to implement error correction, the L2 cache configuration effectively changes to a 1 MB 8-way cache with one cycle added to the latency for cache hits [9].

6. RESULTS

In this section, we present the experimental results for our proposed technique. Section 6.1 compares the cache failure probability for VS-ECC with previously proposed solutions for reducing Vccmin. Section 6.2 evaluates the performance of VS-ECC in the low-voltage mode. Section 6.3 analyzes the power and energy-efficiency of VS-ECC and compares it with previously published techniques.

6.1 Reliability

Figure 6 shows Vccmin results for both persistent failures and soft errors. We use the persistent failure model for 6T cells measured by Kulkarni, et al. [16], in addition to the soft error model proposed by Chishti, et al. [9]. All designs can tolerate a single bit failure due to soft errors or other non-persistent failures, so lines covered by SECDED cannot tolerate any persistent failures. We set a design's Vccmin at a point when 999 out of 1000 caches will not fail due to persistent errors (intersection with the 1.E-03 line in the Figure 1). Our SECDED baseline has a 830 mV Vccmin, below which it does not satisfy the yield requirements. Fixed-strength error correction techniques are successful in lowering the voltage, as more failures can be tolerated: DECTED ECC lowers Vccmin to 675 mV; while 4EC5ED cuts it down to 565 mV. As more error-correction capability is introduced, Vccmin can be lowered even further. However, due to the exponential nature of the bit failure probability curve, we see diminishing returns when we increase error-correction capability beyond 4EC5ED.

Figure 6 also shows that variable-strength ECC can be successful at lowering Vccmin with a fraction of the overhead of fixed-strength solutions. For example, using SECDED on all lines, while using 4-bit correction on only four lines per 16-line set, i.e., VS-ECC-Fixed (12x1, 4x4) reduces Vccmin to 590 mV. VS-ECC-Variable reduces Vccmin to 565 mV (the same as 4EC5ED); while VS-ECC-Disable has a 500 mV Vccmin. VS-ECC-Disable can disable cache lines that have multi-bit failures, and its Vccmin is set at a point to guarantee that we have at least two functioning lines in every set (so we can only disable up to 14 ways in a single L2 set). On average, VS-ECC-Disable only disables 0.4% of all cache lines (i.e., all lines with three or more failures at 500 mV in Figure 1). The best previously published mechanism, MS-ECC [9], achieves a Vccmin of 540 mV at the cost of sacrificing half of the capacity and doubling the number of cache accesses.

6.2 Performance

In this section, we evaluate performance overheads for different cache designs in the low-voltage mode. Figure 7 shows IPC, normalized to the performance of a 2 MB L2 cache baseline with SECDED ECC. For each benchmark category, we show the geometric mean of normalized IPC for the baseline, VS-ECC, and two fixed-strength designs: 4EC5ED and MS-ECC. The 4EC5ED configuration adds a 15-cycle penalty on every cache hit due to the ECC processing, resulting in a 3% drop in IPC on average. MS-ECC, on the other hand, has the same hit latency as SECDED, but also uses half the cache ways to store ECC, resulting in a 50% reduction in size and associativity, and a 6% performance loss. Note that we only show the performance impact of L2 design changes, while keeping the L1 constant.

Compared to the baseline, our VS-ECC designs show a negligible drop in performance (less than 0.1%) across all benchmark categories. This performance loss is significantly lower than that of a fixed-strength 4EC5ED scheme, because VS-ECC-Fixed and VS-ECC-Variable incur the additional 15-cycle penalty only for the 0.1% lines that require multi-bit ECC decoding at 590 and 565 mV (Figure 1). Conversely, all other lines only observe only a single cycle latency for SECDED ECC decoding. VS-ECC-Disable has a negligible performance penalty due to disabling 0.4% of cache lines on average (lines with three or more failures at 500 mV in Figure 1), while a negligible fraction of the remaining lines (less than 3.3%) require multi-bit ECC decoding due to having two persistent failures, or a multi-bit combination of persistent and non-persistent failures. Furthermore, the area overhead of our most area-intensive

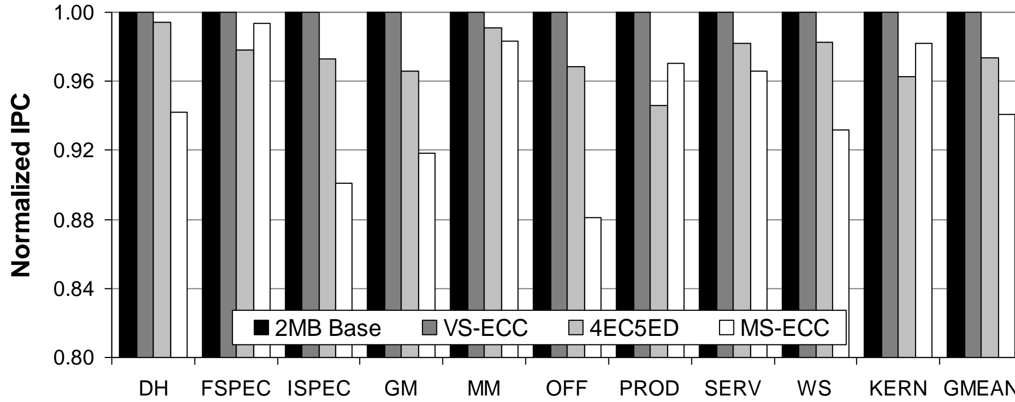

Figure 7. Normalized IPC for different benchmark categories (vs. 2MB Baseline).

Table 3. Energy characteristics of different 2MB cache designs.

Design	Vccmin (mV)	Frequency (MHz)	Norm. Power	Norm. EPI
Baseline (SECDED)	830	2000	1.00	1.000
DECTED	675	1350	0.49	0.722
4EC5ED	565	940	0.26	0.570
MS-ECC	540	830	0.22	0.562
VS-ECC-Fixed	590	1040	0.31	0.587
VS-ECC-Variable	565	940	0.26	0.555
VS-ECC-Disable	500	650	0.16	0.499

implementation, VS-ECC-Variable, represents less than 3% of the total cache area, compared to 1.9% overhead for DECTED and 5.7% for 4EC5ED. Compared to the SECDED baseline, VS-ECC achieves a much lower Vccmin at almost the same performance. Relative to DECTED, VS-ECC achieves a much lower Vccmin and the same performance at a slightly lower area overhead. Compared to 4EC5ED, VS-ECC-Variable achieves the same Vccmin, and VS-ECC-Disable achieves a much lower Vccmin, while getting better performance and using less area.

6.3 Energy Efficiency

Table 3 summarizes the achievable Vccmin, power consumption, energy per instruction (EPI) and energy-delay products of different cache designs during the low-voltage operating mode. We normalize the power and energy results for each design to the baseline, while showing absolute results for Vccmin and frequency. For power calculations, we assume that dynamic power scales quadratically with supply voltage and linearly with frequency. We also assume that static power scales with the cube of supply voltage, and use circuit simulations to model the impact of Vccmin changes on processor frequency [27].

Our VS-ECC-Disable design achieves lower power and lower energy per instruction (EPI) compared to all alternative cache designs. Compared to fixed-strength ECC mechanisms, VS-ECC-Disable is consistently better, reducing power and EPI by 84% and 50%, respectively, vs. SECDED; by 67% and 31%, respectively, vs. DECTED, by 47% and 15%, respectively vs. 4EC5ED; and by 26% and 11%, respectively vs. MS-ECC [9]. Conversely, the recently-published Hi-ECC [26] has a Vccmin of 595 mV on our 2MB baseline cache due to using large 1KB cache lines, but incurs significant additional activity due to read-modify-write operations and reading larger lines that significantly increase dynamic power. Combined with cache line disabling, Hi-ECC achieves the same Vccmin as our VS-ECC-Disable, but requires disabling most cache lines (compared to disabling 0.4% of lines for VS-ECC-Disable with 64B lines).

7. CONCLUSIONS

In this paper, we observe that only a few cache lines experience multi-bit failures at low voltages, while the vast majority of lines exhibit zero or one errors, especially for large caches. We propose a novel cache architecture that relies on variable-strength error-correcting codes (VS-ECC) for error tolerance. The common case of lines with no failures is handled with simple and fast SECDED codes to guarantee recovery from soft errors. A small number of lines with persistent failures can use a strong 4-bit error-correcting code or a variable-length code that requires some additional area and incurs a latency penalty. Each cache set uses a few additional bits to support multi-bit error-correcting codes on a small number of its lines. We also propose a mechanism to dynamically characterize the cache after the processor first transitions to the low-voltage mode to determine which lines will exhibit multi-bit failures and allocate additional ECC bits to them. This algorithm is targeted specifically to alleviate two main drawbacks of online testing: the need for comprehensive testing with pre-set patterns and the need to minimize the impact on end-user experience. Finally, our experimental evaluation demonstrates that compared to fixed-strength ECC techniques, VS-ECC avoids significant decreases in cache capacity, incurs minimal additional area overhead, and avoids unnecessary increases in latency and bandwidth advocated by some current proposals that use large lines. When combined with selective cache line disabling, our VS-ECC design achieves an 84% power reduction and a 50% energy reduction compared to SECDED ECC, and achieves a 26% power reduction and an 11% energy reduction compared to previously published MS-ECC technique, without losing half the cache capacity at low voltages.

8. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. Ilya Wagner was partially supported by the National Science Foundation under Grant #0937060 to the Computing Research Association for the CIFellows Project.

9. REFERENCES

[1] Jaume Abellà, Javier Carretero, Pedro Chaparro, Xavier Vera and Antonio González, "Low Vccmin Fault-Tolerant Cache with Highly Predictable Performance", International Symposium on Microarchitecture, pp. 111-121, Dec. 2009.

[2] Amin Ansari, Shantanu Gupta, Shuguang Feng and Scott Mahlke, "ZerehCache: Armoring Cache Architectures in High Defect Density Technologies", International Symposium on Microarchitecture, pp. 100-110, Dec. 2009.

[3] Elwyn. R. Berlekamp, Algebraic coding theory, New York: McGraw-Hill, chapter 7, 1968.

[4] Hannes Brunner, Andreas Curiger and Max Hofstetter, "On computing multiplicative inverses in GF(2m)", IEEE Transactions on Computers, vol. 42, pp. 1010-1015, Aug. 1993.

[5] Douglas Bossen, Joel Tendler and Kevin Reick, "Power4 System Design for High Reliability", IEEE Micro, vol. 22, No. 2, pp. 16-24, Mar. 2002.

[6] Herbert O. Burton, "Inversionless decoding of binary BCH codes", IEEE Transactions on Information Theory, vol. IT-17, pp. 464-466, 1971.

[7] C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: A state-of-the-art-review", IBM Journal of Research Development, vol. 28, no. 2, pp. 124-134, Mar. 1984.

[8] Robert T. Chien, "Cyclic decoding procedures for Bose-Chaudhuri-Hocquenghem codes", IEEE Transactions on Information Theory, vol. 10, no. 4, pp. 357-363, Oct. 1964.

[9] Zeshan Chishti, Alaa R. Alameldeen, Chris Wilkerson, Wei Wu and Shih-Lien Lu, "Improving Cache Lifetime Reliability at Ultra-low Voltages", International Symposium on Microarchitecture, pp. 89-99, Dec. 2009.

[10] Eiji Fujiwara, Code Design for Dependable Systems: Theory and Practical Applications, Wiley-Interscience, 2006.

[11] A. J. van de Goor, Testing Semiconductor Memories: Theory and Practice, John Wiley & Sons, Inc., NY 1991.

[12] Hideki Imai and Y. Kamiyanagi, "A construction method for double error correcting codes for application to main memories", Transactions of the IECE Japan, vol. J60-D, pp. 861-868, Oct. 1977.

[13] Niraj K. Jha and Sandeep Gupta, Testing of Digital Systems, Cambridge University Press, New York, NY 2002

[14] Jangwoo Kim, Nikos Hardavellas, Ken Mai, Babak Falsafi and James Hoe, "Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding", International Symposium on Microarchitecture, pp. 197-209, Dec. 2007.

[15] Seongwoo Kim and Arun K. Somani, "Area Efficient Architectures for Information Integrity in Cache Memories," International Symposium on Computer Architecture, pp. 246-255, Jun. 1999.

[16] Jaydeep Kulkarni, Keejong Kim and Kaushik Roy, "A 160 mV Robust Schmitt Trigger Based Subthreshold SRAM", IEEE Journal of Solid-State Circuits, vol. 42, no. 10, pp. 2303-2313, Oct. 2007.

[17] James Lee Massey, "Step-by-step decoding of the Bose-Chaudhuri-Hocquenghem codes", IEEE Transactions on Information Theory, vol. 11, no. 4, pp. 580-585, Apr. 1965.

[18] Tomako Matsushima, Toshiyasu Matsushima, and Shigeichi Hirasawa, "Parallel encoder and decoder architecture for cyclic codes", IEICE Trans. on Fundamentals, vol. E79A, no. 9, pp. 1313-1323, 1996.

[19] Saibal Mukhopadhyay, Hamid Mahmoodi and Kaushik Roy, "Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.24, no.12, pp. 1859-1880, Dec. 2005

[20] Thammavarapu R.N. Rao, Eiji Fujiwara, Error-control coding for computer systems, Prentice-Hall, Inc., NJ, 1989.

[21] Irving S. Reed, Ming-Tang Shih, and Trieu-Kien Truong, "VLSI design of inverse-free Berlekamp–Massey algorithm", Proc. IEE Proceedings E on Computers and Digital Techniques, vol. 138, pp. 295-298, Sept. 1991.

[22] David Roberts, Nam Sung Kim and Trevor Mudge, "On-Chip Cache Device Scaling Limits and Effective Fault Repair Techniques in Future Nanoscale Technology", Digital System Design Architectures, Methods and Tools, pp. 570-578, Aug. 2007.

[23] Stefan Rusu, Harry Muljono and Brian Cherkauer, "Itanium 2 Processor 6M: Higher Frequency and Larger L3 Cache", IEEE Micro, vol. 24, No. 2, pp. 10-18, Mar. 2004.

[24] Stanley Schuster, "Multiple Word/Bit Line Redundancy for Semiconductor Memories", IEEE Journal of Solid-State Circuits, vol. 13, no. 5, pp. 698-703, Oct. 1978.

[25] Dmitri Strukov, "The area and latency tradeoffs of binary bit-parallel BCH decoders for prospective nanoelectronic memories," in Asilomar Conference on Signals Systems and Computers, pp. 1183-1187, Oct. 2006.

[26] Chris Wilkerson, Alaa R. Alameldeen, Zeshan Chishti, Wei Wu, Dinesh Somasekhar and Shih-Lien Lu, "Reducing Cache Power with Low Cost, Multi-bit Error-Correcting Codes", International Symposium on Computer Architecture, pp. 83-93, Jun. 2010.

[27] Chris Wilkerson, Hongliang Gao, Alaa R. Alameldeen, Zeshan Chishti, Muhammad Khellah and Shih-Lien Lu, "Trading off Cache Capacity for Reliability to Enable Low Voltage Operation", International Symposium on Computer Architecture, pp. 203-214, Jun. 2008.

[28] Doe Hyun Yoon and Mattan Erez, "Memory Mapped ECC: Low-Cost Error Protection for Last Level Caches", International Symposium on Computer Architecture, pp. 116-127, Jun. 2009.