Reliability-Driven ECC Allocation for Multiple Bit Error Resilience in Processor Cache

Somnath Paul, *Student Member*, *IEEE*, Fang Cai, *Student Member*, *IEEE*, Xinmiao Zhang, *Member*, *IEEE*, and Swarup Bhunia, *Senior Member*, *IEEE*

Abstract—With increasing parameter variations in nanometer technologies, on-chip cache in processor is becoming highly vulnerable to runtime failures induced by "soft error," voltage, or thermal noise and aging effects. Nondeterministic and unreliable memory operation due to these runtime failures can be addressed by: 1) designing the memory for worst-case scenarios and/or 2) runtime error detection and correction. Worst-case guard-banding can lead to overly pessimistic results for cell footprint and power. On the other hand, conventional error correcting code (ECC) used in processor cache has very limited correction capability, making it insufficient to protect memory in scaled technologies (sub-45 nm), which are vulnerable to multiple-bit failures in a word (64-bit). The requirement to tolerate multibit failures is accentuated with supply voltage scaling for low-power operation. We note that due to inter and intra-die parameter variations, different memory blocks move to different reliability corners. A uniform ECC protection for all memory blocks fails to account for the distribution of vulnerability across memory blocks. On the other hand, it can lead to overly pessimistic results if the worst-case vulnerability of a memory block is accounted for during ECC allocation. In this paper, we propose a reliability-driven ECC allocation scheme that matches the relative vulnerability of a memory block (determined using postfabrication characterization) with appropriate ECC protection. We achieve postfabrication variable ECC allocation by storing the check bits in the "ways" of an associative cache. We use shortened Bose-Chaudhuri-Hocquenghem (BCH) cyclic code with zero padding, which provides high random error correction capability with modest amount of check bits. Moreover, we propose efficient circuit/architecture-level optimizations of the ECC encoding/decoding logic to minimize the impact on area, performance, and energy. Simulation results for SPEC2000 benchmarks show that such a variable ECC scheme tolerates high failure rates with negligible performance (*four percent*) and area (*0.2 percent*) penalty.

Index Terms—Cache, runtime failures, soft error, process variation, variable ECC allocation.

1 Introduction

Tolerance to runtime failures in large on-chip caches has emerged as one of the key challenges in present day microprocessor design [13]. Such failures can affect contiguous bit positions—e.g., soft errors in static random access memory (SRAM) cells caused by striking of high-energy alpha and neutron particles [6], [7]. On-chip caches are also becoming increasingly vulnerable to random errors [4]. These errors can be caused by: 1) noise on the supply voltage line [5], 2) thermal noise, or 3) temporal degradation due to aging effects [9]. The situation is exacerbated due to increasing process variations in nanoscale CMOS technologies, which makes a memory cell vulnerable to different forms of parametric failures [2].

Runtime errors have been conventionally addressed using either parity or low-cost Hamming codes, which are capable of single error correction and double error detection (SECDED) [19]. However, simple SECDED protection is proving insufficient [11], [12] due to multiple-bit upsets (MBUs) in a codeword [7], [8]. To address MBUs, a combination of SECDED and bit interleaving has emerged as the widely accepted choice [27]. Bit interleaving distributes the contiguous errors into different words and facilitates correction using SECDED. However, it comes at significant energy overhead [11] due to reads performed on unwanted words. On the other hand, existing approaches for multiple-bit random failure tolerance either address only static (not runtime) failures during manufacturing test [12] or can substantially compromise storage space in the memory [3].

Due to inter and intra-die process variations, different sections of a memory array may move to different process corners [10]. This leads to a distribution of vulnerability to runtime failures across memory blocks in different chips and within a chip. This is illustrated in Fig. 1. Sizes for these blocks typically range from 1 to 8 KB. Simulations with a 2 MB processor cache in 45 nm technology shows 7-10X variations in reliability across memory blocks. A uniform ECC protection for all memory blocks fails to account for the distribution of vulnerability across blocks and can provide overly pessimistic results if ECC allocation is based on the worst-case vulnerability of the blocks.

In this paper, we propose a variable ECC allocation scheme, which assigns correction capability to individual memory blocks based on their relative vulnerabilities to runtime failures. This is achieved by incorporating ECC encode/decode logic with varying error correction capability during design and selecting them on demand during actual operation. A major challenge with multiple bit error correction is that increasing error correction capability significantly enhances the number of extra check bits due

• *The authors are with the Department of Electrical Engineering and Computer Science, Case Western Reserve University, 10900 Euclid Avenue, Cleveland, OH 44106.*
E-mail: {sxp190, fxc72, xinmiao.zhang, skb21}@case.edu.

Manuscript received 25 Jan. 2010; revised 7 June 2010; accepted 29 Aug. 2010; published online 23 Sept. 2010.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-2010-01-0056.
Digital Object Identifier no. 10.1109/TC.2010.203.

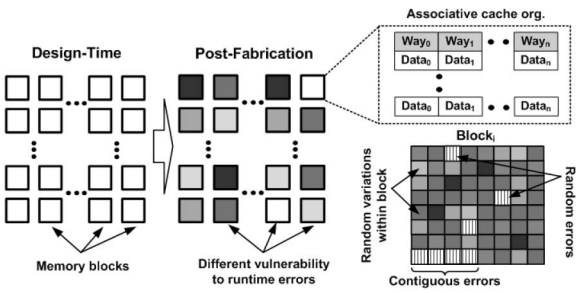

Fig. 1. After fabrication, blocks of a cache can move to different reliability corners due to inter-die and systematic intra-die variations. Conventionally, caches are set-associative which are organized into multiple "ways." Runtime failures in a block can result from either random or contiguous failures.

to ECC. Besides, variable ECC allocation based on reliability of a block would require a programmable storage to store the check bits. We address this problem by storing the check bits in the "ways" of an associative cache similar to [3]. Sacrificing one or more ways in a subset of memory blocks incurs modest performance loss ($\sim$4 percent). The ECC protection is achieved in a way that does not affect memory cell design, and hence, integration density.

We note that effectiveness of a multibit error tolerance scheme largely depends on choice of ECC and efficient design of the encode/decode logic. To this end, we propose using shortened BCH cyclic code [30] with zero padding for variable ECC protection and a circuit/architecture codesign approach for low-overhead implementation of encode/decode logic. We explore a fast BCH implementation, which exploits the property of systematic codes and performs syndrome detection in one processor clock cycle and encoding in two cycles to minimize performance overhead. The correction logic is invoked only when the decoding logic detects any error, which provides the opportunity to power-gate the correction logic most of the time for saving power. To further minimize the area and power overhead due to ECC encode/decode logic, we propose effective sharing of hardware resources among encode/decode logic of multiple ECC blocks. The proposed approach also enables dynamically updating ECC of a memory block to adapt to scaled operating voltage for low-power operation. It also allows adaptation to temporal reliability degradation due to various device aging effects [9]. The paper analyzes the impact of the proposed variable ECC scheme on low-power reliable memory operation and provides extensive simulation results for benchmark applications to demonstrate the effectiveness of the approach.

The rest of the paper is organized as follows: Section 2 presents the background for this work. Section 3 describes the proposed variable ECC scheme. Section 4 provides the rationale behind the choice of BCH code and its low-overhead implementation. Section 5 presents simulation results on failure tolerance capability and power performance overhead. Section 6 describes ways to minimize overhead. Section 7 concludes the paper.

2 Background

Runtime failures in processor cache can be classified as: 1) random failures due to voltage/thermal noise and 2) contiguous failures due to soft error. Although the Soft Error Rate (SER) of an SRAM cell is found to remain constant across process technologies, due to the increase in the memory size per generation of the chip, the total number of soft errors increases with processor generation [6], [7]. Moreover, as the SRAM footprint decreases with process scaling, the probability of MBUs due to a single-particle strike increases [7], [8]. Memory cells can also experience failures due to aging-induced temporal degradation. Aging effects such as Bias Temperature Instability (BTI) [9] can considerably degrade the read stability for SRAM over time leading to read failures.

Runtime failures in processor caches have been conventionally addressed by ECC. The level of ECC protection is typically different for L1 and L2/L3 caches due to their different design criteria. L1 caches are typically designed for higher performance and reliability. In order to minimize the performance overhead due to ECC, simple parity check is conventionally employed in L1. Large last level caches such as L2 or L3 are conventionally protected using SECDED ECC scheme [11], [12]. However, the increased vulnerability of memory cells to runtime failures, specially under low-power operating modes, has drawn considerable attention to multiple bit error tolerance for on-chip caches [11], [12]. Existing techniques to tolerate multiple-bit failures in processor cache during execution can be classified as: 1) architectural techniques [13], which reduce the probability of multiple bit errors by periodic writeback of the dirty data to the next level of memory hierarchy, and 2) ECC-based techniques [3], [11], [12] to tolerate multiple-bit failures. The former class of approaches cannot prevent corruption of memory content when a multibit random or contiguous error happens. The latter class of approaches is more effective in tolerating multiple errors and several salient approaches have been proposed. The scheme proposed in [11] can tolerate large number of clustered failures using 2D (in both X and Y directions) error coding with simple SECDED codes combined with bit interleaving. However, the approach incurs large power and performance overhead for random errors, which limits its applicability [12]. The scheme proposed in [12] attempts to correct multiple-bit random errors in L2 cache. It leverages on the access locality of the L2 to store the predecoded ECC bits for sections of the cache which are accessed more frequently. However, the scheme is limited to only 2-bit error correction and 3-bit error detection (DECTED). Moreover, due to the use of additional storage for storing the extra parity bits and minimizing the decoding latency, the scheme incurs almost 36 percent L2 energy overhead. A recent work [3] aims at providing higher ECC protection to on-chip cache during low-voltage operation. The solution uses Orthogonal Latin Square Coding (OLSC) as ECC and devotes one or more cache ways for storing the ECC check bits. It allocates fixed ECC to each ECC memory block, and hence, cannot prevent pessimistic ECC allocation that chooses ECC for worst-case block reliability.

The proposed approach differs from the earlier works in several ways. First, it proposes a "variability-aware" ECC allocation, which matches the reliability of a block with

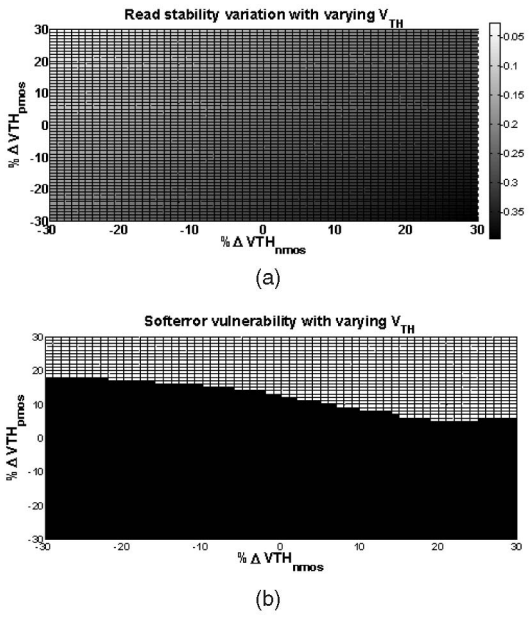

Fig. 2. Impact of process variation on the vulnerability of SRAM cell toward runtime failures. (a) Memory blocks which have moved to a strong-NMOS weak-PMOS corner are more prone to read failures. (b) Cells with weak PMOS are more prone to soft-error-induced failures.

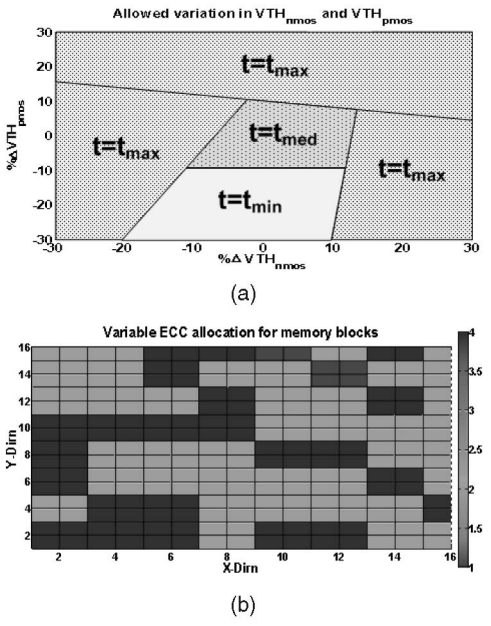

Fig. 3. (a) Based on variability, the spectrum of NMOS and PMOS variation is divided into multiple ECC regions. (b) ECC allocation for memory blocks on a die at worst-case inter-die corner.

appropriate level of ECC protection, and hence, prevents pessimistic fixed ECC as in [3]. Variable ECC allocation makes it amenable for adaptation to nonuniform temporal within-die variation in block reliability due to aging effects. Second, the proposed approach uses BCH coding. Although the correction latency for OLSC [3] is less than that in BCH, the number of check bits in OLSC is significantly worse compared to BCH for the same correction performance. As a result, for the same level of ECC protection, the scheme in [3] sacrifices much larger section of the cache to store ECC bits leading to large IPC overhead ($\sim$10 percent). In addition, the proposed approach employs circuit architecture co-optimization of encoder/decoder hardware to further minimize the performance and power overhead. Third, it explores application of the variable ECC scheme for tolerating both random and contiguous runtime errors in a unified error protection framework.

3 Reliability-Driven Variable ECC Allocation

In this section, we study the impact of process variation on runtime failures and present the architectural considerations to realize variable ECC implementation for large on-chip L2/L3 caches.

3.1 Impact of Parameter Variations on Runtime Failures

Due to process parameter variations, memory cells which are marginally functional during manufacturing test can undergo runtime failures due to voltage/thermal noise, soft error, or aging effects. These cells (referred to as "weak" cells [14], [15) must, therefore, be protected using more ECC compared to cells which have suffered little variations. As argued in [16], memory cells which have either moved to the low threshold (V_t) or the high V_t region due to inter/intra-die variations are susceptible to read, write, access, and hold failures. We observe a similar trend in our simulations (Fig. 2). An SRAM cell at 45 nm was simulated in HSPICE using PTM45nm LP models [17]. The threshold voltage for both the NMOS and the PMOS transistors in the SRAM cell was varied by $+/-30$ percent of the nominal V_t and the subsequent effect on read stability and soft-error tolerance for the cell was noted. The read stability values from our simulation are plotted in Fig. 2a. It is defined by $V_{trip} - V_{read}$, where V_{trip} denotes the threshold voltage for the inverter storing logic "1" in the SRAM cell and V_{read} is the voltage pulse that is generated at the node storing logic "0" in the SRAM cell. From Fig. 2, we note that a weak-PMOS and a strong-NMOS make the SRAM cell most vulnerable toward flipping during the read operation. In addition to block-level inter/intra-die corners, cells within a particular block can suffer from random local variations. The effect of random intra-die variations is aggravated depending on inter and systematic intra-die corner in which a particular memory block lies [16].

The soft-error immunity for the SRAM cell is severely degraded when the PMOS becomes weak under inter/intra-die variation. In Fig. 2b, light color denotes all the cells which have flipped due to a charged particle strike. For this simulation, we have adopted the soft-error model as presented in [28]. On the basis of the scaling trends presented in [28], critical charge for SRAM cell at 45 nm node was estimated to be 1 fC. We note that more recent works [33] also place the value of Q_{crit} to be between 1 and 2 fC for scaled technologies. The value of the fall time constant is taken to be 5 ps [29].

Based on the above analysis, we note that a $+/-10$ percent ΔV_t variation from the nominal value is sufficient to cause the cell to be more vulnerable to one or more modes of runtime failures. As shown in Fig. 3a, the

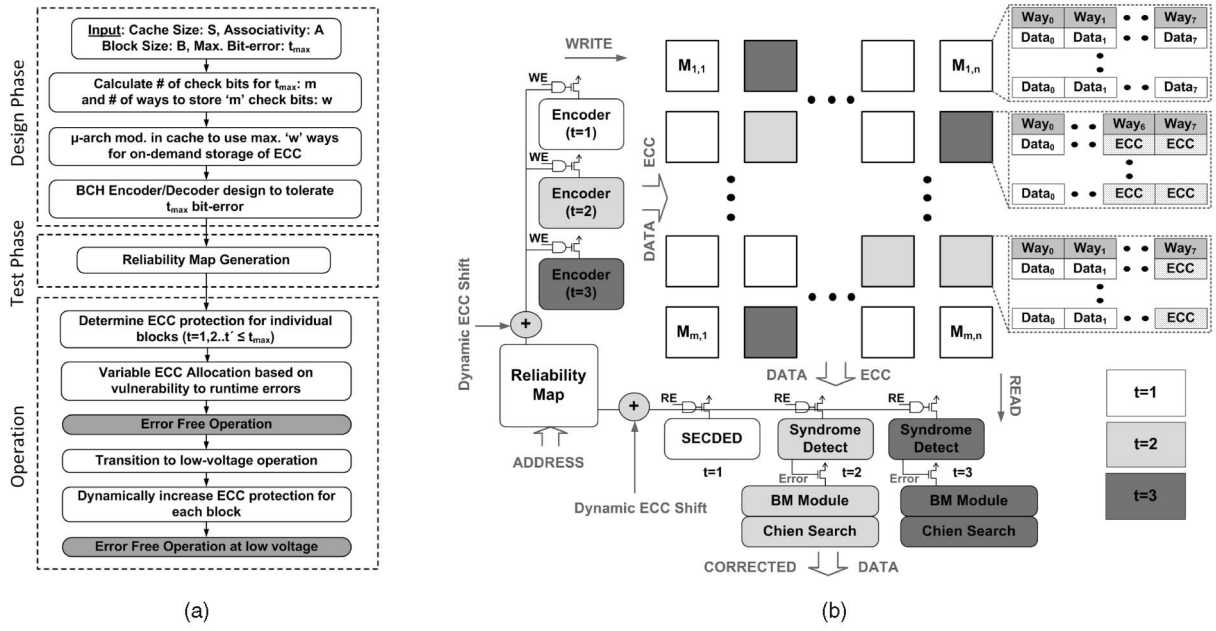

Fig. 4. (a) Major steps in variable ECC allocation. (b) Architecture for postfabrication variable ECC allocation based on the process corner of the individual memory blocks.

entire spectrum for PMOS and NMOS V_t variation in an SRAM cell can be divided into three regions based on the extent of ECC protection they should receive. We note that a strong PMOS and a nominal NMOS ($\Delta V_t < +/-10$ percent of the nominal value) are most resilient to the failure mechanisms considered above. Cells with such configuration should, therefore, be assigned the least ECC protection ($t = t_{min}$, where t denotes the number of errors that may be corrected for a given block). Cells with PMOS and NMOS V_t variation less than $+/-10$ percent of the nominal value should be assigned a higher ECC protection ($t = t_{med}$) due to their vulnerability toward soft error. Cells falling outside the above ΔV_t regions should be assigned the maximum ECC protection ($t = t_{max}$) to tolerate runtime failures arising from system noise, soft errors, and aging effects.

In order to explore the merits of variable ECC assignment for multibit runtime failure tolerance, we simulated a 2 MB L2 cache model. The model is divided into 256 memory blocks, each with size of 8 KB. One thousand instances of the same model were simulated considering an inter-die process variation with $\sigma_{inter} = 25\%$. For each instance, a systematic intra-die variation with $\sigma_{sys} = 20\%$ and random intra-die variation with $\sigma_{intra} = 15\%$ was also considered. Based on the V_t distribution of the memory blocks, variable ECC was then assigned to the memory blocks for each instance of the L2. Fig. 3b shows the ECC assignment for the instance which has the least number of memory blocks with $t = t_{max}$. From this experiment, we infer that a postfabrication variable ECC allocation cannot only correct the vulnerability of different memory blocks within a single die, but also allows appropriate ECC allocation to different dies based on their specific inter-die corners.

Since the proposed variable ECC assignment is performed postfabrication, it necessitates the generation of a reliability map during manufacturing test. We note that number of techniques [1], [14], [15], [16] have already been proposed to generate postsilicon reliability map for embedded memories. Since read, write, and access failures are accelerated under lower supply voltage and/or higher operating temperature, a March test performed on the memory array at multiple operating conditions can effectively reveal the reliability of the individual blocks [15], [16]. In [16], the authors have proposed to determine the inter-die process corner of a memory chip by monitoring the leakage current of a sufficiently large SRAM array. Once the PMOS and NMOS corner is determined during manufacturing test, the corresponding information can be stored on a small on-chip memory to be used during postfabrication ECC assignment.

3.2 Overall Scheme

Fig. 4a shows the major steps of the proposed variability-aware ECC allocation methodology in large caches. As seen in Fig. 4a, the cache size (S), cache associativity (A), block size (B), and the maximum number of bit errors that can be tolerated (t_{max}) serve as input specifications for the variable ECC allocation methodology. During the design phase, based on t_{max}, the number of required check bits (m) and the number of ways (w) required to store m check bits are calculated. Suitable μ-arch modifications to interchangeably use these "w" ways for data and ECC storage are then incorporated into the cache. Finally, a low-latency and energy-efficient BCH encoder/decoder is implemented at the memory interface to detect and correct up to t_{max} bit errors. On postfabrication during manufacturing test, a reliability map is generated which characterizes the vulnerability of different sections of the cache to runtime errors. The baseline vulnerability is determined by the amount of parametric variation (measured in terms of read/write margins) present in the most reliable memory block. The baseline vulnerability is used to determine the value of t_{min}.

Based on the reliability map, values of $t_{min} < t <= t_{max}$ for other memory blocks are determined by their relative vulnerabilities to parametric failures. For variable ECC allocation during deployment, these t values are encoded into 2-bits and are stored on chip in a small nonvolatile memory. Based on these encoded values, variable number of ways in each set are dedicated to storage of ECC bits by asserting appropriate select signals to the ECC selection logic. During runtime, the ECC selection logic is responsible for distinguishing the data from ECC bits.

Fig. 4b shows the corresponding L2 cache architecture to enable the proposed variable ECC allocation. For a given error rate, let the three levels of ECC protection assigned to a L2 cache be $t_{min} = 1$, $t_{med} = 2$, and $t_{max} = 3$. $t = t_{min} = 1$ denotes the scenario where a single error in a memory word (32-b or 64-b) can be safely corrected using conventional SECDED scheme. Typically, the SECDED protection is applied to the L2 data arrays, while runtime failures in the smaller tag arrays are typically avoided by either conservative design of the tag array or simple parity assignment [18]. Assuming eight ECC bits per 64-b of L2 cache, SECDED scheme for 2MB L2 cache will require a storage of 256 KB. Conventionally, a separate storage is dedicated for storing these ECC bits on chip [19]. In the variable ECC assignment scheme, multiple bit error correction facility in less reliable memory blocks is achieved using BCH codes. For example, $t = 2$ denotes a 2-b error correction and 4-b error detection scheme. Higher values of t require extra ECC bits. For example, for each 64-b word, the number of ECC bits required for $t = 2$ and $t = 3$ are 14 and 21, respectively. In order to accommodate these extra ECC bits, we use one or more of the cache ways to store ECC bits in place of actual data bits, similar to [3]. This is illustrated in Fig. 4b. Let us assume a 8-way L2 cache, with each way storing 64B of data. If the number of ways required to store ECC bits is denoted as w and if the number of ECC bits required to correct a t bit error in a 64-b of data is denoted as m, then w can be easily calculated from the following relation: $(8 - w) \times m \times 8 - (8 - w) \times 8 \times 8 < 64 \times w \times 8$.

If the number of ways devoted to ECC storage is "w," then the number of ways storing data is 8-w. So, the number of check bits required for all the data bits is (8-w)×m. Since there are eight segments in each 64-B word, the total number of check bits is $(8\text{-w}) \times \text{m} \times 8$. The SECDED storage can store eight check bits for each 64-b of data, and hence, $(8\text{-w}) \times 8 \times 8$ check bits for $64 \times (8\text{-w})$ bytes of data stored in (8-w) data ways. So, the number of check bits which actually needs to be stored in cache ways is $(8\text{-w}) \times \text{m} \times 8 - (8\text{-w}) \times 8 \times 8$. This must be smaller than the total storage in "w" cache ways that is devoted to ECC ($64 \times \text{w} \times 8$). Since the value of "m" is known for a given "t," the value of "w" can be calculated from the equation. From the above expression, we see that for $t = 2, m = 14$, $w = 1$ and for $t = 3, m = 21$, $w = 2$.

The proposed scheme exploits the fact that modern L2/L3 caches typically have large number of "ways," some of which can be used to store check bits for on-demand error tolerance. For a "fast" cache implementation [24], both data and check bits can be read out from the data array, selected on the basis of way hit, and then, passed through the syndrome detection logic to detect any error. However, for a "normal" cache implementation, reading out the data and the ECC bits simultaneously would require additional ECC signals (28 additional bits to allow 4-bit error correction in a 64-bit word) inside the cache. Another possible solution is to allow an additional cycle for reading out the ECC bits following the data read out. Thus, contrary to the 2D error coding scheme proposed in [11], allocating one or more ways in the same row for ECC storage does not cause unnecessary read from multiple rows. Similar to an only SECDED scheme, any store shorter than the ECC word (64-b) requires a read-modify-write operation to the ECC stored in the way as well as to the ECC bits in the memory conventionally used for storing SECDED. Decoding for error correction involves two components: 1) logic for detection of error and 2) finding out the error location in order to achieve error correction. For realistic error rates, error-free read will be performed most of the time. Since the detection logic appears in the critical path for L2 read, we have implemented a single-cycle decoding logic for BCH. The correction circuit is invoked only if an error is detected and is pipelined allowing a 64-b throughput per clock cycle. This simple architecture allows a postfabrication ECC allocation based on the runtime failure rate.

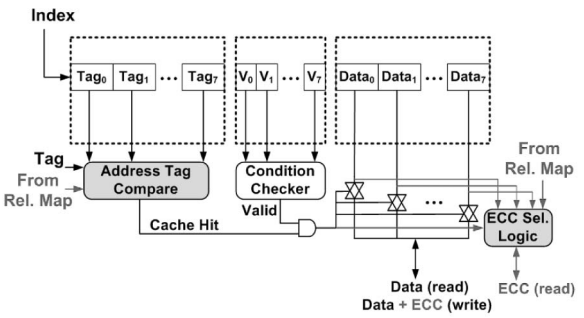

Fig. 5. μ-architecture showing procedure to store and retrieve ECC bits from one or more cache ways.

3.3 μ-Architectural Modifications for Variable ECC Allocation

Fig. 5 shows the modifications to the L2 cache μ-architecture required to realize the variable ECC assignment. In addition to the address tag, an enable signal from the reliability map would now serve as an input to the *Address Tag Compare* logic (Fig. 5). Given the memory blocks are classified into four or less ECC bins, this signal would be only 2-b wide. This actually serves to invalidate the tag comparison operation for one or more ECC ways so that during read, data out is not selected from any of these ways. If there is a hit in any of the remaining data ways, the cache hit signal and the enable signal coming from reliability map can be used to separate the ECC bits from the data bits using an *ECC selection logic*, as shown in Fig. 5. During write, the ECC bits stored in these ways are modified based on the incoming data to be written. Write occurs to the cache way for which the tag hit has occurred.

In evaluating the performance for variable ECC, we note the number of runtime errors in the L2 cache that are either transferred to the processor pipeline during loads or are transferred to the main memory in case of a read miss or a write miss. Fig. 6 shows the L2 states during which the encoder, error detection, and correction logic are invoked.

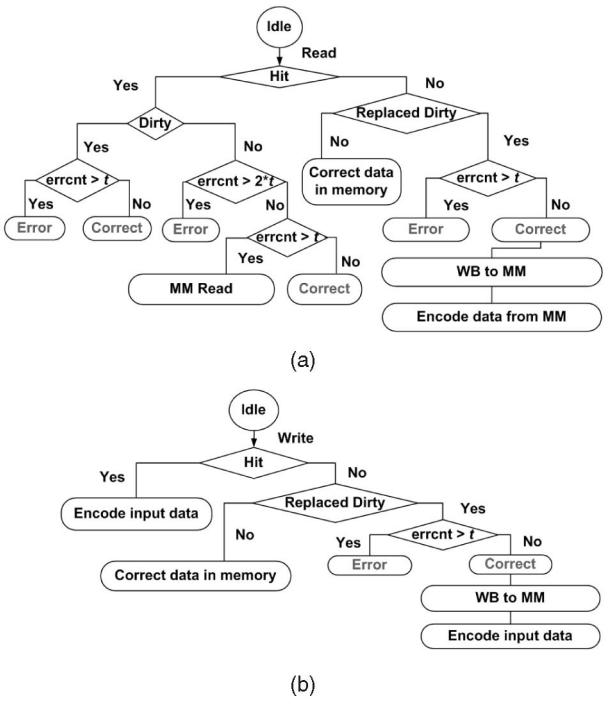

Fig. 6. States of the L2 ECC controller for error detection and correction.

Here, we assume a write-back L2 with a write-allocate scheme. As we note from Fig. 6, a read request can result in a hit or a miss. For a hit, the memory word can be dirty or nondirty. If it is a dirty block and the number of errors exceed the maximum number of errors that can be corrected for the block, then we have an error in the system. If the number of bits in error is less than t, the errors can be detected and corrected using the detection and correction logic, respectively. For a hit to a nondirty block, if the number of errors is greater than $2 \times t$, then we have an error. If, however, the number of errors lies in the range $t < e <= 2 \times t$, then the detection logic will be able to detect these errors in the memory word. If the block is clean, the correct data can then be brought into the L2 from the main memory. On a read miss, a valid entry that is being replaced can again be dirty or clean. If the block being replaced is dirty and the number of errors in the dirty block is less than t, then error correction is performed. Write miss follows the same sequence of actions as read miss. On a write hit, however, only the encoding logic is invoked.

4 Energy-Efficient Implementation of ECC

For multiple bit error correction, we propose to adopt BCH as the error correcting code. BCH code is a large class of powerful random error-correcting cyclic codes [30]. A BCH code is uniquely defined by its code length n and number of information bits k. A code that can correct t bits errors (detecting $2t$ bits errors) is called (n, k, t) code. The implementation of BCH encoder/decoder for the proposed multibit error tolerance scheme requires special considerations. They are as follows:

- For L2, writes are not as frequent as reads, and therefore, moderate increase in write latency due to encoding of data blocks does not affect the overall performance significantly. Employing BCH for error correction in on-chip caches, therefore, requires a fast encoding/decoding approach. Since runtime errors are infrequent, the correction latency can be allowed to be much higher than the error detection latency. With these relative constraints, we implemented a BCH-based multiple bit error correction logic such that latency for error detection is least ($\sim 1\ cycle$), encoding requires $\sim$2-3 $cycles$ and latency for error correction is $\sim 20\ cycles\ or\ more$, depending on the maximum number of errors that can be corrected by the code.
- In our specific application, the length of information bits is $k = 64$ and the number of correctable bits can vary from $t = 1$ to $t = 6$. However, BCH code is conventionally of length $2^p - 1$ and the number of information bits (k) varies with respect to t (refer to Table 1a). Therefore, in order to implement a BCH code where the number of information bits is fixed and the code length varies with respect to t, we employ a scheme known as shortening of the BCH code (refer to Table 1b) [30]. Code shortening essentially sacrifices the code rate to maintain the equivalent correction capability. The shortening scheme can be better explained by an example. For instance, to reduce a BCH code from a primitive (127,120,1) form to a shortened (71,64,1) form, we could interpret that the 64-bit length message vector m is appended by $120 - 64 = 56$ bit of zeros. It is to be noted that the appended 0 bits have no effect on the syndrome computation.
- The area overhead for BCH encoder, decoder, and correction logic for multiple values t can be quite substantial. We, therefore, explore an area-efficient BCH implementation which shares hardware among the encoder, decoder, and correction logic for multiple values of t.
- In order to minimize the power overhead for the correction logic, the correction logic is invoked only if an error is detected by the detection logic. At all other times, the correction logic remains supply gated to save leakage power.

TABLE 1
Primitive and Shortened BCH Codes:
(a) Primitive and (b) Shortened

n	k	t
127	120	1
127	106	3
127	85	6

(a)

n	k	t
71	64	1
85	64	3
106	64	6

(b)

The rest of this section describes the energy-efficient BCH implementation with the above considerations.

4.1 Encoding Logic

The standard sequential encoding circuit for cyclic code requires the cycles to be scaled with the length of information bits. Even with the unfolding technique, the encoding latency

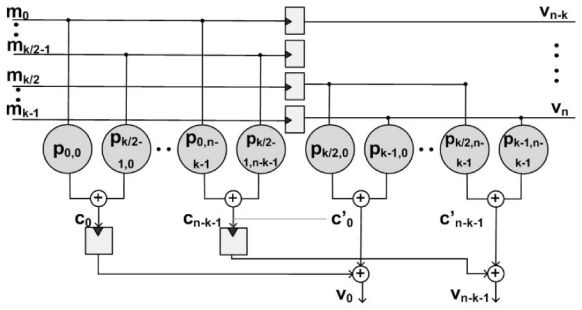

Fig. 7. Pipelined encoder logic.

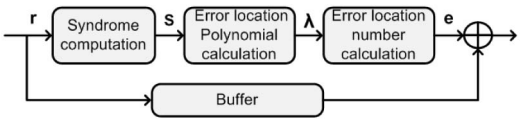

Fig. 8. BCH decoder architecture.

is still at the level of $O(k)$. Thus, in order to meet the above latency requirements, encoding in our scheme is achieved by exploiting the systematic property for BCH codes. In the proposed BCH encoding scheme, the code vector v is directly obtained by the following matrix multiplication of the message bits m and the generator matrix G:

$$v = m * G. \tag{1}$$

Fig. 7 shows the basic architecture of encoding circuit. In Fig. 7, $p_{i,j}$ are the elements of the G matrix and denotes a connection if $p_{i,j} = 1$ and no connection if $p_{i,j} = 0$. The adder denotes a modulo-2 adder. The encoding operation is pipelined into two stages.

4.2 Syndrome Detection and Correction Logic

As illustrated in Fig. 8, decoder for the BCH is divided into three parts:

1. Syndrome Computation,
2. Determination of the error location polynomial, and
3. Determination of the actual error locations.

As suggested by [30], on dividing the received vector $r(x)$ by the relevant minimal polynomial associated with each row of parity check matrix H, the remainder $b(x)$ has a degree much less than $r(x)$. However, the computation of $b(x)$ requires a sequential structure similar as serial encoding which, in general, requires n cycles. Therefore, to minimize the latency, a direct approach is employed to accomplish syndrome computation in one cycle. This approach to generate the syndrome vector (S) using matrix multiplication similar to the encoding step is shown below:

$$S = r * H^T. \tag{2}$$

The basic architecture for the detection logic is shown in Fig. 9. It is very similar to the encoding logic as shown in Fig. 7, only difference being the $p_{i,j}$ values and generation of the final *syndrome detect* output. As seen from Fig. 9, syndrome detect signal is $2 \times t$ bits, t being the maximum number of errors that can be corrected by the BCH encoding.

In step 2 of BCH decoding, computation of the error location polynomials is facilitated by inverse-free

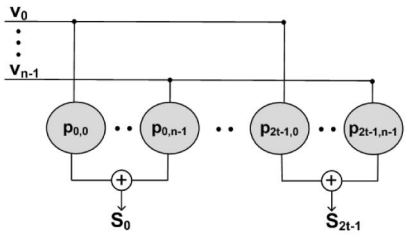

Fig. 9. Syndrome detection logic.

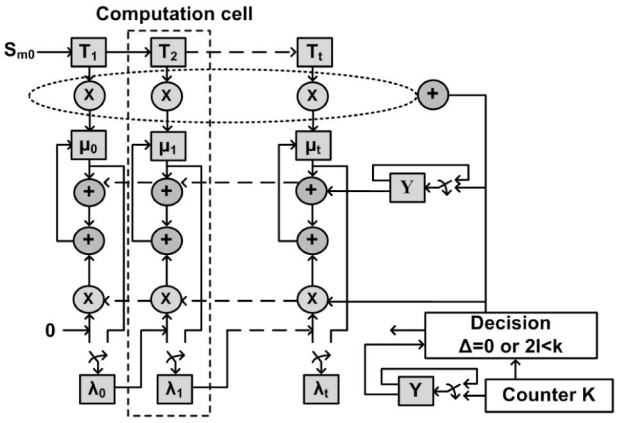

Fig. 10. Implementation of the Berlekamp-Massey algorithm.

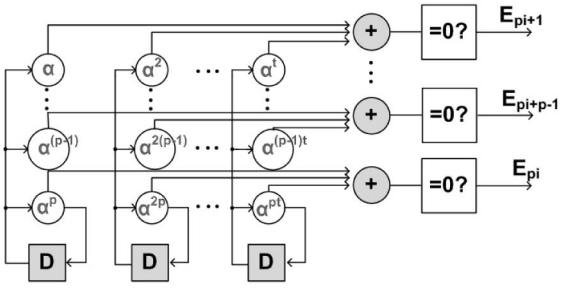

Fig. 11. Implementation for Chien search algorithm.

Berlekamp-Massey algorithm from IBM [31]. This step requires $2 \times t$ number of cycles, where t refers to the maximum number of errors that can be corrected by the code. We follow the highly regular and homogeneous architecture as proposed in [31] for our implementation. Fig. 10 shows the architecture of Berlekamp-Massey algorithm. As illustrated in Fig. 10, the circuit is composed of a collection of standard computation cells. The number of standard computation cells is equal to t.

In step 3, Chien search is used to evaluate the error location numbers. The latency of polynomial computation is $2 \times t$. A serial implementation of Chien search module would require k cycles, which for a 64-b ECC word would be 64 clock cycles. For higher bit error rates where correction is frequent, this would lead to a significant overhead in L2 read latency. A parallel structure is, therefore, implemented to reduce the decoding cycles to k/p. The architecture implemented is adopted from [32]. Fig. 11 shows the parallel implementation for the Chien Search module. For our implementation, the value of the parallelization factor p is four.

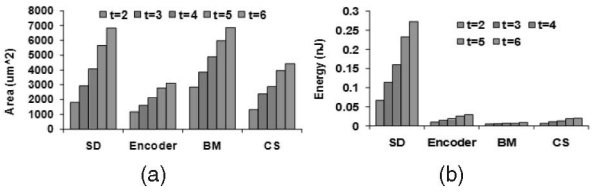

Fig. 12. Synthesis results showing (a) area and (b) energy requirement for the proposed BCH implementation (SD: Syndrome Detect, BM: Berlekamp-Massey, CS: Chien Search).

4.3 Hardware Results

Design overhead for the ECC logic is obtained by synthesizing the designs in Synopsys Design Compiler (DC) for a 45 nm technology node. Figs. 12a and 12b show the area and energy requirement for the encoding and decoding logic. As expected, with higher correction capability, the area and energy requirement increases. In order to compare the area between our variable ECC logic and L2, area required by a 2 MB eight-way L2 at 45 nm node was estimated using Cacti [24]. If design time provision is made for ECC logic with correction capability corresponding to $t = 2, 3, 4$, the total area required by all modules of the ECC logic is only *0.18 percent* of L2 cache area. For $t = 4, 5, 6$ version, total ECC area is only *0.3 percent* of L2 cache area. These results show that the ECC logic introduced for multiple bit error correction is negligible compared to the area of the L2 cache.

5 Simulation Setup and Results

The effectiveness of the proposed ECC allocation scheme was validated for the following scenarios:

1. tolerance to random failures;
2. tolerance to contiguous failures;
3. simultaneously tolerating both random and contiguous failures; and
4. tolerating high failure rates at low voltage.

The proposed ECC allocation scheme was evaluated for a 2 MB eight-way L2 cache (with each way storing 64B of data). The L2 cache access latency was assumed to be 12 clock cycles. Configuration parameters for the baseline processor were adopted from [22]. L2 cache for the baseline processor was assumed to be protected by an SECDED-only scheme. Power and performance impact for the proposed ECC allocation scheme was observed for SPEC2K benchmark suite [20] compiled for 64b Alpha processor and simulated using Simplescalar toolsuite [21]. Each of the SPEC benchmarks was simulated for 100M instructions with 500M instructions fast forwarded. For evaluating the tolerance to runtime random errors, a weighted random pattern generation tool was integrated with the Simplescalar setup to insert random runtime errors into the L2 cache following a given bit failure probability (BFP). A BFP of 10^{-n} indicates a single-bit failure out of 10^n bits. Considering a wide range of device parameter variations for PTM 45 nm model, a large BFP range (10^{-7}-10^{-4}) was observed in our SPICE simulations. We note that a similar range has been observed in [3]. At lower supply voltages, P_{fail} is much higher (10^{-4}-10^{-3}). We have, therefore, validated the effectiveness of the proposed approach for this wide range of BFP. In our simulations, the Mean Time To Failure (MTTF) is taken to be 10^5 cycles. Hence, the error pattern is regenerated every 10^5 cycles. Note that the choice of MTTF has been made to facilitate the simulation process. A smaller or larger value of MTTF would not affect the average number of failures in a simulation window, and hence, is not expected to affect the choice of ECC or the energy/performance trends. Energy values for the baseline processor and the proposed variable ECC scheme are obtained for 100 nm CMOS process using Wattch [23].

5.1 Tolerance to Random Errors

For a given BFP, relative vulnerabilities of cache blocks to runtime failures determine the maximum number of bit errors (t_{max}) that must be detected and corrected to ensure error-free operation in the cache. Here, we estimate the values of t_{max} required at different BFP values for both worst- and best-case reliability maps.

5.1.1 $BFP = 10^{-7}$

For a runtime failure rate as low as 10^{-7}, we assume that in the proposed scheme, a variable ECC allocation with $t_{min} = 1, t_{med} = 2$ and $t_{max} = 3$ is being used. For $BFP = 10^{-7}$, all the failures in a particular word were found to be less than or equal to 1 and could be corrected by a simple SECDED scheme. For BFP $= 10^{-7}$, the proposed scheme, therefore, incurs higher energy and performance overhead compared to a simple SECDED-based correction scheme. Total energy overhead is calculated considering the number of reads, writes, and corrections performed corresponding to the $t = 2$ and $t = 3$ blocks and the energy estimates obtained for the synthesized encoder, detector, and the correction logic. From Fig. 13, we note that although the average performance and energy overhead are negligible for the proposed scheme, L2 miss rate increases due to higher conflict misses in $t = 2$ and $t = 3$ memory blocks.

5.1.2 $BFP = 10^{-6}$

For $BFP = 10^{-6}$, Table 2 shows the number of failures for the individual benchmarks in the baseline case. As evident from Table 2, for the proposed scheme with $t = 1$, 2, and 3 distribution, failures cannot be eliminated for all the benchmarks. In order to guarantee a failure-free operation for BFP $= 10^{-6}$ in the worst-case reliability scenario, the only solution is to move to a $t = 2$, 3, and 4 distribution with $t_{min} = 2$. For the die with the best reliability map, having blocks with $t = 1$, 2, and 3 protection suffices to ensure a failure-free operation. As evident from Table 2, the downside of $t = 2$, 3, and 4 distribution is the increase in the L2 miss rate. The CPI overheads for variable ECC allocation for the worst and best reliability maps for no failures are *1.84* and *0.92 percent*, respectively. Corresponding energy overheads are *0.91* and *0.93 percent*, respectively. Results for individual benchmarks are shown in Fig. 14.

5.1.3 $BFP = 10^{-5}$

For BFP $= 10^{-5}$, the baseline processor with simple SECDED protection experiences more failures. Number of failures over the course of 100M instructions is showed in Fig. 15a. However, in the proposed scheme, for both worst and best reliability maps, an ECC distribution of $t = 2$, 3

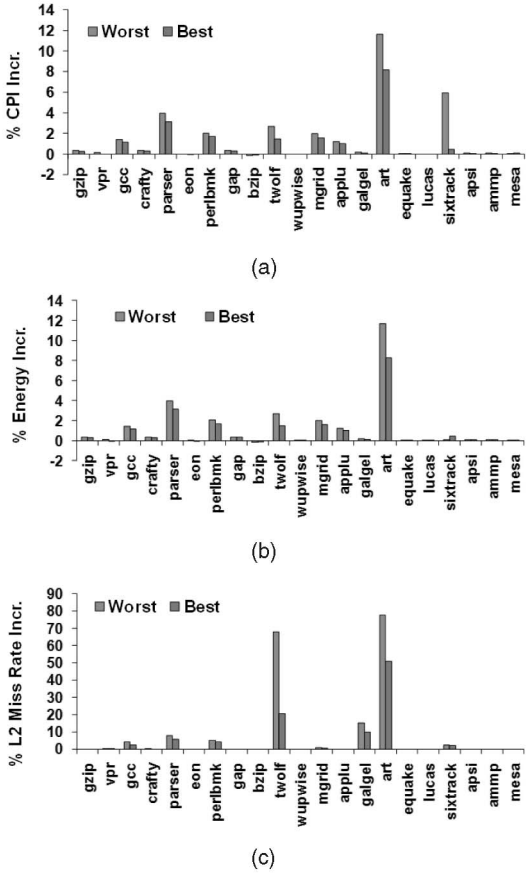

Fig. 13. Percentage increase in (a) CPI, (b) energy, and (c) L2 miss rate for the proposed scheme with $\mathrm{BFP} = 10^{-7}$ for worst and best-case reliability maps.

and 4 is able to guarantee an error-free operation. This tolerance, however, comes at CPI and energy overheads as reported in Figs. 15b and 15c. Average CPI overheads for the worst- and best-case scenarios are *2.01* and *1.73 percent*, respectively. Similarly, the average energy overheads are *2.03* and *1.75 percent*, respectively. Average increase in the L2 miss rate are *16.4* and *14.05 percent*, respectively.

5.1.4 $BFP = 10^{-4}$

For $\mathrm{BFP} = 10^{-4}$, the baseline processor suffers from significant number of runtime failures for both worst- as well as best-case reliability maps (refer to Fig. 16). For the proposed scheme, a $t = 2$, 3, and 4 distribution is able to correct *99.9* and *99.8 percent* of all the failures that occur in the baseline processor. In order to ensure an error-free operation, all the memory blocks must be protected with $t_{min} = 4$ for the die with the worst-case and $t_{min} = 3$ for the die with best-case reliability maps, respectively. When the proposed scheme with $t_{min} = 4$ is used to protect all memory blocks in a die with worst reliability map, the average increase in CPI and energy are *3.58* and *3.8 percent*, respectively. For the best reliability map, the CPI and energy increase are *2.47* and *2.53 percent*, respectively. Average increase in L2 miss rate are *17.14* and *17.12 percent*, respectively (Fig. 17).

Since 100M instructions is a short window in the life of a program, we have also validated the effectiveness of the proposed methodology for 1B instructions in a $BFP = 10^{-4}$

TABLE 2
ECC Allocation for $\mathrm{BFP} = 10^{-6}$

Benchmark	# of Failures for Worst Map		% L2 Miss Rate Incr.	
	Baseline	Proposed ($t_{min} = 1$)	Worst	Best
gzip	0	0	0.00	0.00
vpr	0	0	0.12	0.06
gcc	0	0	6.90	2.49
crafty	0	0	0.71	0.47
parser	0	0	12.20	5.79
eon	0	0	0.00	0.00
perlbmk	0	0	8.94	4.47
gap	0	0	-0.18	-0.53
bzip	0	0	-0.06	-0.06
twolf	1	0	170.11	20.69
wupwise	0	0	0.00	0.00
mgrid	0	0	1.03	0.61
applu	1	0	0.00	0.00
galgel	3	0	24.37	10.08
art	1	1	115.34	50.96
equake	0	0	0.00	0.00
lucas	0	0	0.00	0.00
sixtrack	0	0	4.29	2.00
apsi	0	0	0.00	0.00
ammp	0	0	0.00	0.00
mesa	0	0	0.10	0.00
Average			**16.38**	**4.62**

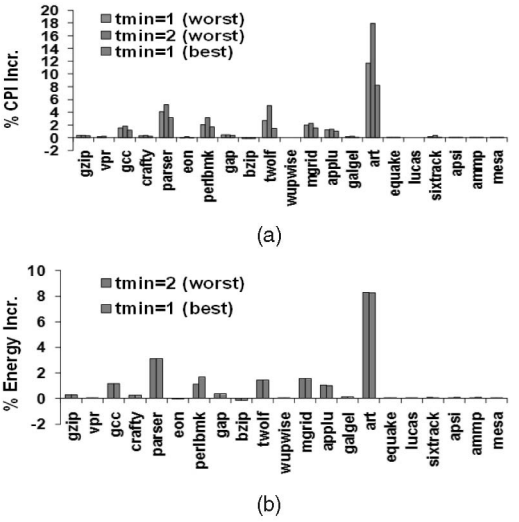

Fig. 14. Percentage increase in (a) CPI and (b) energy for the proposed

scenario. Fig. 18 shows that the average CPI and energy overheads with the proposed error tolerance scheme are *4.42* and *4.47 percent*, respectively. This is *0.8* and *0.9 percent* higher compared to the results obtained for 100M instructions. The nominal overhead numbers for 500M instruction fast forwarding and 1B instruction simulation suggest that the proposed error tolerance scheme would continue to remain effective at acceptable CPI and energy overheads for varied sampling windows.

5.1.5 Comparison with Earlier Works

A number of earlier works [11], [12] have tried to address multiple bit error correction in cache using on-chip ECC. We compare our scheme with the multiple bit error correction methodology as presented in [11]. The 2D error coding employs either an interleaved "xor" or an SECDED scheme in

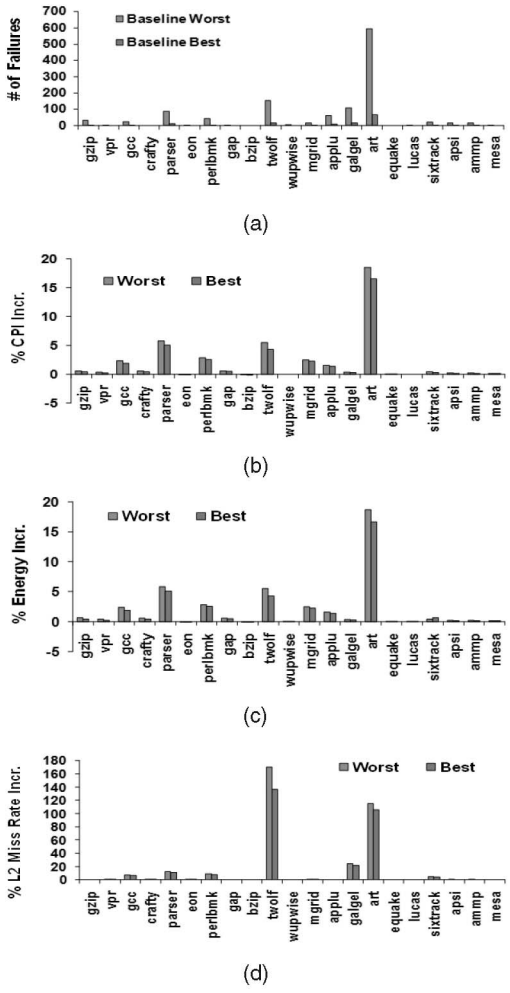

Fig. 15. (a) Number of failures in the baseline processor with worst and best reliability maps. Percentage increase in (b) CPI, (c) energy, and (d) L2 miss rate for the proposed scheme with $\mathrm{BFP} = 10^{-5}$.

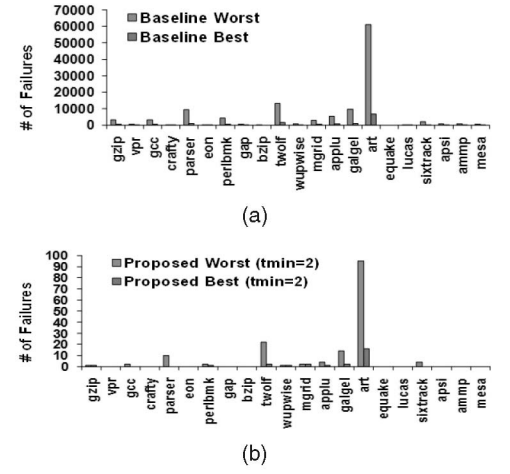

Fig. 16. Number of failures in (a) baseline and (b) proposed scheme with $\mathrm{BFP} = 10^{-4}$.

the horizontal direction. In the vertical direction, the coding applies an interleaved "xor"-based parity calculation. However, due to a SECDED-based protection scheme, more than 2 bits of error in the same memory word goes undetected. For

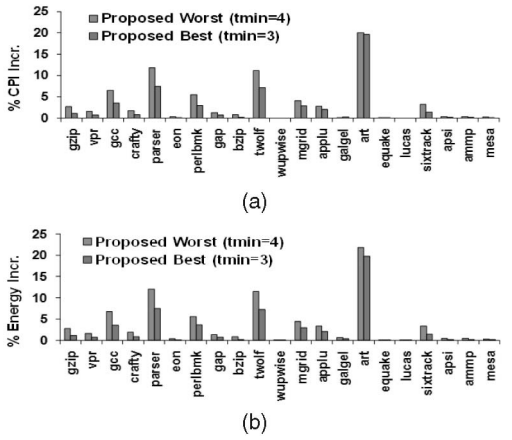

Fig. 17. Percentage increase in (a) CPI and (b) energy for the proposed scheme with $\mathrm{BFP} = 10^{-4}$.

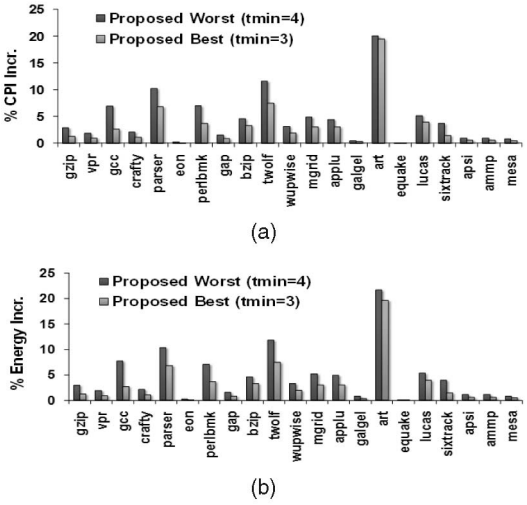

Fig. 18. Percentage increase in (a) CPI and (b) energy for the proposed scheme with $\mathrm{BFP} = 10^{-4}$ for 1B instructions.

2-bit error detected in the horizontal direction, correction using vertical code can still fail if the errors in the interleaved rows form the vertices of a rectangle, as shown in Fig. 19a. Fig. 19b shows for $BFP = 10^{-4}$, the number of failures that cannot be tolerated by [11] when both horizontal and vertical words are protected by simple parity (EDC). The number of errors reduces when an SECDED scheme is used in the horizontal direction but is still nonzero. The 2D error coding scheme also incurs considerable energy and performance overhead for tolerating random errors. Each write requires reading from eight interleaved vertical rows. The energy overhead due to additional reads and bit interleaving is significant. Fig. 19c reports the energy overhead for tolerating random errors with error rate 10^{-4}. On average, increase in energy for $BFP = 10^{-4}$ is *8.65 percent*. However, the proposed scheme tolerates all errors with only *3.8 percent* increase in energy.

5.2 Tolerance to Soft-Error-Induced Contiguous Errors

For protection against soft-error-induced failures, we assume that the baseline processor is protected using SECDED along with four-way bit interleaving. Such a

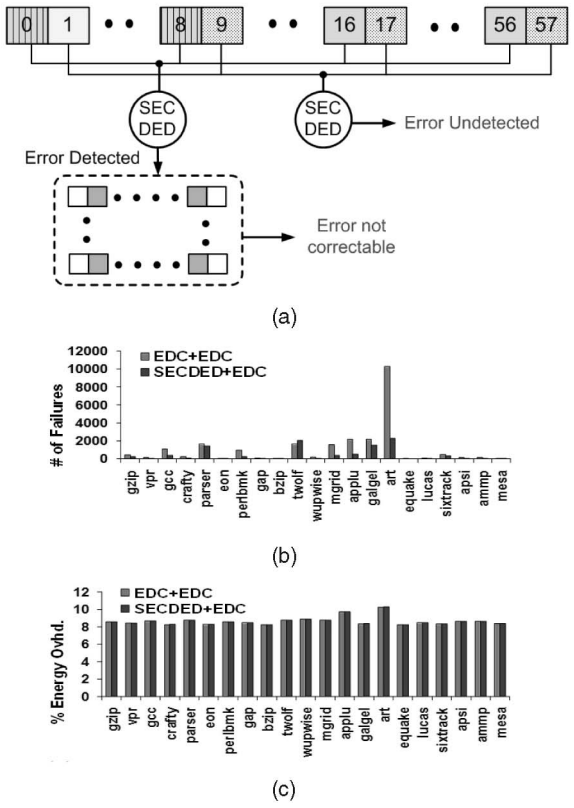

Fig. 19. (a) Error scenarios for the 2D-coding-based multiple bit error tolerance scheme as presented in [11]. (b) Number of failures that cannot be tolerated by 2D error coding for $\mathrm{BFP} = 10^{-4}$. (c) Percentage increase in energy for 2D-coding-based multiple bit error tolerance.

system can correct up to four contiguous errors and detect up to eight. This suffices to cover the maximum number of soft-error-induced contiguous failures (4, 5, or 7) corresponding to $N_c = 16$, 32, or 64, where N_c refers to the number of memory cells between two well taps [8]. However, as pointed out in [11], bit interleaving comes at additional power, delay, and area overhead. Power overhead in an interleaved cache, which occurs primarily due to access of unwanted words in the same row, has been shown to be more than *100 percent* of a noninterleaved one [11]. It is, therefore, interesting to see the trade-off involved in tolerating multiple bit errors using on-chip BCH-based ECC as opposed to bit-interleaving.

For memory designs with $N_c = 16$ or 32, an ECC distribution with $t = 4$, 5, and 6 suffices to cover the maximum number of contiguous errors that may occur. Penalty is, however, paid in terms of higher conflict miss and increased CPI overhead since 2-3 ways in each set are used for storing the ECC instead of data. Fig. 20 shows the CPI overhead for the individual benchmarks for worst- and best-case reliability maps. The average CPI overhead in these two cases are *3.52* and *3.29 percent*, respectively. Although the L2 miss rate increases significantly, the variable ECC allocation actually leads to energy savings over the baseline processor which employs bit interleaving for tolerating multiple-bit contiguous errors. Total energy consumption for the baseline processor was estimated assuming the fact that bit interleaving doubles the energy

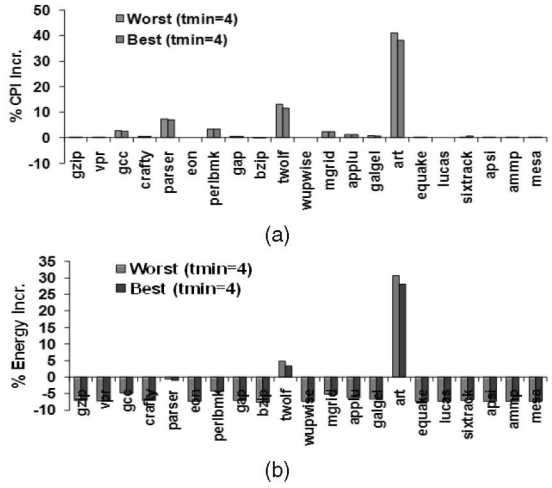

Fig. 20. Percentage increase in (a) CPI and (b) energy when the proposed scheme is used for soft-error tolerance with $t_{min} = 4$.

requirement for each L2 access [11]. For the proposed scheme without bit interleaving, energy calculation considers the overhead due to $t = 4$, 5, and 6 ECC logic. Fig. 20b shows the energy overhead for the individual benchmarks. Average savings for the worst and best maps are *4.33* and *4.13 percent*.

5.3 An Unified Solution to Tolerate Random and Contiguous Errors

Here, we discuss the trade-offs involved with schemes that can serve as unified solutions to tolerate both random and contiguous errors in memory words.

5.3.1 $t_{min} = 4$, 5, and 6 Distribution with Extra Storage

Variable ECC allocation with $t = 4$, 5, and 6 distribution leads to inordinate increase in L2 miss rate for a set of benchmarks. This can be addressed if additional storage is available for storing the extra parity bits required by BCH encoding rather than using one or more ways to store the ECC bits. Assuming that an additional storage for SECDED is already present, storage required for storing 20 extra parity bits corresponding to $t = 4$ is calculated to be *31 percent* of the L2 area. For blocks with $t = 5$ and $t = 6$, we assume that the additional parity bits occupy one out of eight ways in L2. This area increase is with respect to a L2 cache without bit interleaving scheme. If this area overhead can be tolerated, then the average L2 miss rate increase is a minimal *3.68 percent*. CPI overhead for the worst reliability map, as shown in Fig. 21a, is also negligible due to lower conflict misses, average increase being only *0.75 percent*. In spite of the extra energy expended in reading from and writing to this additional memory, this scheme leads to overall energy savings. Read/write energy at 100 nm technology node for this additional memory was estimated using Cacti [24]. After accounting for this additional energy overhead, a $t = 4$, 5, and 6 distribution with extra memory leads to an average energy savings of *6.64 percent*. Results for individual benchmarks are shown in Fig. 21b.

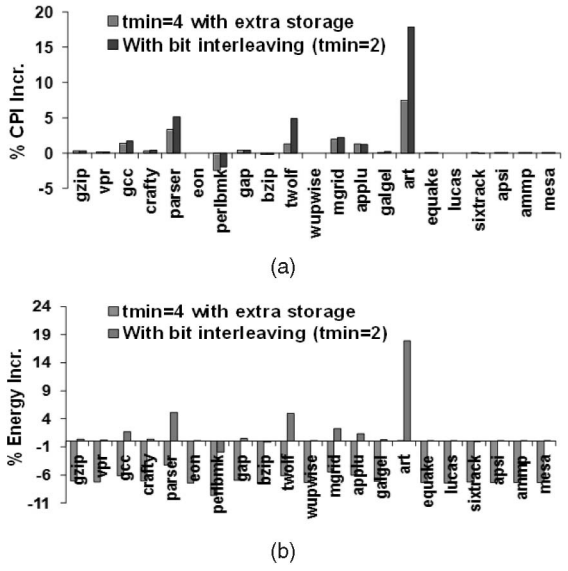

Fig. 21. Overhead in (a) CPI and (b) energy when the proposed scheme with $t_{min} = 2$ is used along with bit interleaving for soft-error tolerance.

5.3.2 $t_{min} = 2$, 3, and 4 Distribution with Bit Interleaving

A variable ECC allocation with $t = 2$, 3, and 4 and four-way bit interleaving tolerates both: 1) random errors with error rate as high as 10^{-4} and 2) up to 8-bit contiguous errors in $t = 2$ blocks and up to 16-bit contiguous errors in $t = 4$ blocks, making it amenable to tolerance of high soft error rate and scalable to future technology nodes. With respect to a L2 cache without bit interleaving, this scheme does not incur any extra area overhead since extra ECC bits for $t = 2$, 3, and 4 are stored in ways. The average CPI overhead is only *1.56 percent* (refer to Fig. 21a). With respect to an only bit-interleaved L2, this scheme incurs an average *1.57 percent* increase in energy requirement (refer to Fig. 21). The average L2 miss rate, however, increases by *11.64 percent*.

5.3.3 Comparison of the Unified Schemes

Table 3 lists the area, performance and energy overhead, percentage of successful correction, and L2 miss rate increase for the following schemes:

- $t = 4, 5, and\ 6$ without any additional storage for the ECC bits.
- $t = 4, 5, and\ 6$ with additional storage for the ECC bits.
- $t = 2, 3, and\ 4$ with four-way bit interleaving.

Percentage of successful correction was estimated by noting the number of corrected errors out of the total number of errors for a soft error rate of 1,000 FIT/Mb [6]. The percentage was calculated for three different values of N_c with maximum errors being 4, 5, and 7, respectively. Scheme 1 experiences high L2 miss rate and also fails to provide complete correction for all values of N_c. Scheme 2 has the lowest L2 miss rate at the cost of high area penalty. However, it fails to provide complete correction for larger values of N_c. But schemes 1 and 2 provide energy savings over the baseline processor employing a four-way bit interleaving scheme. Scheme 3 incurs minimal CPI and energy overheads over the bit-interleaved baseline configuration. It can, however, tolerate higher number of contiguous errors. Depending upon design constraints, any of the above schemes can, therefore, be used to tolerate both types of runtime errors.

TABLE 3
Comparison of Different ECC Schemes

Scheme	Avg CPI. Ovhd.	Avg. Energy Incr. (%)	Area Incr. (%)	L2 Miss Incr. (%)	% of Successful Correction N_c		
1	3.5	+4.3	0	60.4	100	99.3	77.3
2	0.8	-6.6	31	3.7	100	99.4	77.3
3	1.6	-1.6	0	11.6	100	100	100

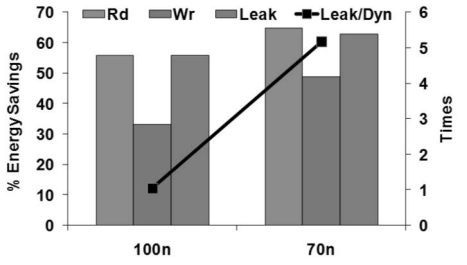

Fig. 22. Percentage savings due to voltage scaling in read, write, and leakage power of an SRAM cell at 100 and 70 nm technology nodes.

5.4 Energy Saving with Dynamic Voltage Scaling

The proposed multibit error tolerance scheme provides an opportunity to dynamically allocate ECC for the memory blocks. Such dynamic adaptation can be used to achieve power saving by combining voltage scaling for the memory cells with higher ECC protection. Consider that the nominal L2 operates under an error rate of 10^{-5}. As mentioned earlier, a $t = 2$, 3, and 4 distribution is followed to ensure error-free operation for the L2. The nominal operating voltage for the memory cell at 100 nm technology node is chosen to be 1 V. When the supply voltage is reduced by 200 mV, the read stability of the SRAM cell reduces by $3\times$ of its value at nominal operating conditions. Since the effect of process variations and voltage/thermal noise is more severe under low operating voltages, we assume that the effective BFP during low-power operating mode is 10^{-4}. A 200 mV reduction in supply voltage reduces the read, write, and leakage powers by *55.86*, *33.15*, and *55.7 percent*, respectively.

This energy saving due to low-power operation outweighs the overhead in providing extra ECC protection for $BFP = 10^{-4}$. Using Cacti, we first estimate the ratio of leakage to dynamic power for a 2 MB cache. As shown in Fig. 22, for 100 nm node, this ratio is *1.03*. The total energy consumption including the leakage overhead for the L2 is then estimated for both nominal as well as low-power operating conditions. After considering the energy overhead for $t = 4$ ECC protection for all memory blocks, average energy savings at 100 nm node with the proposed scheme is calculated to be *5.93 percent* (Fig. 23). Higher savings in energy is achieved for 70 nm technology node. At 70 nm node, savings in read, write, and leakage energy increases to *64.76*, *48.77*, and *62.8 percent*, respectively (refer to Fig. 22). The ratio of leakage to dynamic energy also

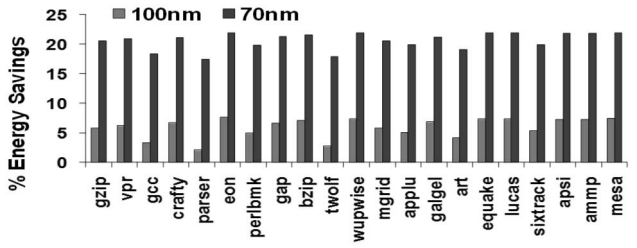

Fig. 23. Percentage energy savings at 100 and 70 nm technology nodes when the on-chip L2 cache is operated in a low-power mode with the variable ECC scheme accounting for the increased failure rate in low-power mode.

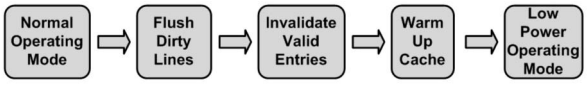

Fig. 24. Steps for on-chip cache to transition from normal to low-power operating mode before their ECC protection can be dynamically increased.

increases to *5.17*. Taking into account all these factors, the average energy savings at 70 nm is *20.58 percent* (Fig. 23). Note that in order to provide increased protection during low-power operation, the cache must go through a number of transitions before it fully operates in the low-power mode. These actions, as illustrated in Fig. 24, ensures that correct ECC bits are stored for the memory blocks. Without flushing of dirty lines and invalidation of the valid cache entries, some cache blocks will continue to hold ECC bits corresponding to $t = 2$ although the syndrome detection will be performed for $t = 4$ corresponding to $BFP = 10^{-4}$. For long duration of low-power operation, the transition overhead can be quickly amortized.

The ability to dynamically adapt the ECC protection for a block also helps to tolerate increased error rate due to temporal degradation mechanisms such as BTI and other aging effects on demand. Aging effects often lead to nonuniform variations in reliability across blocks. The proposed variable ECC scheme is amenable for adaptation to the aging-induced within-die reliability distribution.

6 Design Considerations and Extensions

The reliability map is essential to implement the proposed reliability-driven ECC allocation scheme. If the map is generated only once during manufacturing test, one-time programmable read-only memory (ROM) can be used for this purpose. If on-chip cache requires to be calibrated at runtime, a multiple-time programmable nonvolatile memory (NVM), such as embedded flash, can be used to store the reliability map. Considering a 2 MB cache, which is divided into 8 KB memory blocks and the reliability value for each block is encoded into 2 bits, the total storage requirement for the reliability map is only 64 Bytes. Since the effectiveness of the variable ECC scheme rests on the correct reliability map information, it is essential to prevent its corruption. Hence, the reliability map needs to be conservatively designed and sufficiently protected with ECC, as proposed earlier in [1] for protecting the defect map.

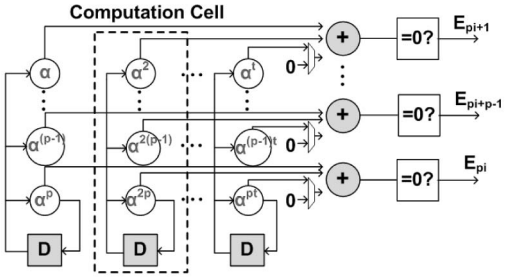

Fig. 25. Configurable Chien search module implementation allowing hardware sharing across different values of t.

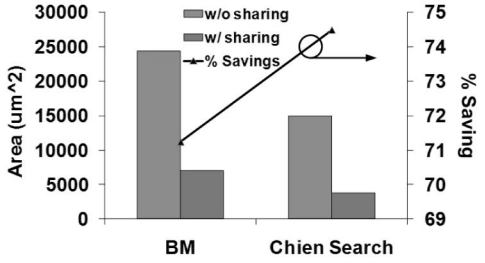

Fig. 26. Percentage area savings with BM and Chien search module implementations that allow hardware sharing.

6.1 Hardware Sharing to Reduce Overhead

The encoder and syndrome detection logic has structural similarity since both involve operations with matrices with constant coefficients. Common Subexpression Elimination (CSE) approach [26] can be used to identify the common patterns in the G and H matrices and share them. For the BM architecture illustrated in Fig. 10, since the number of computation cells is equal to t, the designer can keep provision for the highest value of t to be supported for L2. Later, based on the cache block being corrected, appropriate number of cells can be enabled. For example, if 2-bit correction is supported for a block, only first $t = 2$ computation cells are enabled. The input of the data need to be switched to corresponding T_i according to different values of t. Suppose the maximum capability of error correction t is 6, then for any $t' \leq t$, the data are loaded to $T_{6-t'+1}$ and the computation cells indexed from 1 to $6 - t'$ are disabled. Fig. 25 shows the architecture to achieve hardware sharing in the Chien search module. If $t_{max} = 6$, the $t = 2$ code only requires the first three computation cells and rest four cells can be disabled. Fig. 26 shows that at 45 nm technology node, percentage area savings for these modules are *71.24* and *74.48 percent*, respectively, with such sharing.

6.2 Address Remapping and Dirty Bit Protection

A major drawback of using one or more ways for storing the ECC bits instead of data bits is higher conflict miss for some applications. The increase is due to the fact that for the worst-case reliability map, the most frequently accessed memory addresses for these applications are mapped at most vulnerable blocks. Since most of these applications enjoy high locality of reference, it is possible to map the most accessed memory addresses to more reliable blocks, thereby reducing the conflict miss. Such preferential mapping has already been investigated in the context of tolerating parametric failures and hard defects [25]. Moreover, for higher BFP, a significant percentage of the total

failures arise due to insufficient dirty bit protection. If the address profile for an application is known beforehand, then exploiting the locality of access, the proposed scheme can provide higher ECC protection to the memory space that contains maximum number of dirty bits.

7 Conclusion

Tolerance to runtime failures, both random (e.g., due to thermal/voltage noise) and contiguous (e.g., due to soft error) have become a major challenge for large on-chip caches fabricated using nanoscale technologies. Worst-case design of memory or conventional ECC schemes cannot provide sufficient protection against these failure modes, while maintaining integration density, energy, and performance. We have presented an efficient variable ECC scheme that exploits the distribution of memory block reliability under inter and intra-die process variations. For a specific bit error rate, the proposed scheme allocates ECC of varying error correction capability to different memory blocks based on their relative vulnerability to failures. Area, performance, and energy overhead for the proposed scheme are minimized by appropriate choice of ECC and joint circuit/architecture-level optimizations of the encoding/decoding hardware. In contrast to existing multiple bit error tolerance schemes, the proposed approach can tolerate both random and contiguous errors. We have also analyzed the effectiveness of the approach in dynamic voltage scaling for low-power memory operation, which requires higher ECC protection at scaled voltage. The proposed scheme can be combined with bit-interleaving policy to enhance the on-chip error correction capability manifold. Future investigation will include application of reliability-aware address mapping and extra protection for dirty blocks to further reduce the overhead of ECC.

References

[1] A. Agarwal, B.C. Paul, H. Mahmoodi, A. Datta, and K. Roy, "A Process-Tolerant Cache Architecture for Improved Yield in Nanoscale Technologies," *IEEE Trans. Very Large Scale Integration Systems,* vol. 13, no. 1, pp. 27-38, Jan. 2005.

[2] S. Mukhopadhyay, H. Mahmoodi, and K. Roy, "Modeling of Failure Probability and Statistical Design of SRAM Array for Yield Enhancement in Nanoscaled CMOS," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 24, no. 12, pp. 1859-1880, Dec. 2005.

[3] Z. Chisti, A.R. Alameldeen, C. Wilkerson, W. Wu, and S. Lu, "Improving Cache Lifetime Reliability at Ultra-Low Voltages," *Proc. Int'l Symp. Microarchitecture,* 2009.

[4] C. Constantinescu, "Trends and Challenges in VLSI Circuit Reliability," *IEEE Micro,* vol. 23, no. 4, pp. 14-19, July/Aug. 2003.

[5] M. Agostinelli et al., "Erratic Fluctuations of SRAM Cache V_{min} at the 90nm Process Technology Node," *Proc. Electron Devices Meeting,* 2005.

[6] C.W. Slayman, "Cache and Memoryc Error Detection, Correction, and Reduction Techniques for Terrestrial Servers and Workstations," *IEEE Trans. Device and Materials Reliability,* vol. 5, no. 3, pp. 397-404, Sept. 2005.

[7] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong, "Characterization of Multi-Bit Soft Error Events in Advanced Srams," *Proc. Int'l Electron Devices Meeting,* 2003.

[8] K. Osada, K. Yamaguchi, and Y. Saitoh, "SRAM Immunity to Cosmic-Ray-Induced Multierrors Based on Analysis of an Induced Parasitic Bipolar Effect," *IEEE J. Solid-State Circuits,* vol. 39, no. 5, pp. 827-833, May 2004.

[9] K. Kang, S. Gangwal, S.H. Park, and K. Roy, "NBTI Induced Performance Degradation in Logic and Memory Circuits: How Effectively Can We Approach a Reliability Solution?" *Proc. Asia and South Pacific Design Automation Conf.,* 2008.

[10] M. Mutyam et al., "Process Variation-Aware Adaptive Cache Architecture and Management," *IEEE Trans. Computers,* vol. 58, no. 7, pp. 865-877, July 2009.

[11] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J.C. Hoe, "Multi-Bit Error Tolerant Caches Using Two-Dimensional Error Coding," *Proc. Int'l Symp. Microarchitecture,* 2007.

[12] H. Sun, N. Zheng, and T. Zhang, "Leveraging Access Locality for the Efficient Use of Multibit Error-Correcting Codes in L2 Cache," *IEEE Trans. Computers,* vol. 58, no. 10, pp. 1297-1306, Oct. 2009.

[13] S.S. Mukherjee, J. Emer, T. Fossum, and S.K. Reinhardt, "Cache Scrubbing in Microprocessors: Myth or Necessity?," *Proc. Int'l Symp. Dependable Computing,* 2004.

[14] D.M. Kwai et al., "Detection of SRAM Cell Stability by Lowering Array Supply Voltage," *Proc. Asian Test Symp.,* 2000.

[15] A. Pavlov et al., "Weak Cell Detection in Deep-Submicron SRAMs: A Programmable Detection Technique," *IEEE J. Solid-State Circuits,* vol. 41, no. 10, pp. 2334-2343, Oct. 2006.

[16] S. Mukhopadhyay, K. Kim, H. Mahmoodi, and K. Roy, "Design of a Process Variation Tolerant Self-Repairing SRAM for Yield Enhancement in Nanoscaled CMOS," *IEEE J. Solid-State Circuits,* vol. 42, no. 6, pp. 1370-1382, June 2007.

[17] Predictive Technology Models (PTM), http://www.eas.asu.edu/ptm/modelcard/LP/45nm LP.pm, 2010.

[18] S. Rusu, H. Muljono, and B. Cherkauer, "Itanium 2 Processor 6M: Higher Frequency and Larger L3 Cache," *Proc. Int'l Symp. Microarchitecture,* 2004.

[19] J.L. Shin, B. Petrick, M. Singh, and A.S. Leon, "Design and Implementation of an Embedded 512-KB Level-2 Cache Subsystem," *IEEE J. Solid State Circuits,* vol. 40, no. 9, pp. 1815-1820, Sept. 2005.

[20] SPEC CPU 2000 Benchmarks, http://www.spec.org/cpu2000/, 2010.

[21] Simplescalar Toolset V3.0, http://www.simplescalar.com/, 2010.

[22] H. Li et al., "DCG: Deterministic Clock-Gating for Low-Power Microprocessor Design," *IEEE Trans. Very Large Scale Integration,* vol. 12, no. 3, pp. 245-254, Mar. 2004.

[23] Wattch v1.02d, http://www.eecs.harvard.edu/dbrooks/wattch-form.html, 2010.

[24] Cacti v4.1, http://www.hpl.hp.com/research/cacti/, 2010.

[25] M. Mutyam and V. Narayanan, "Working with Process Variation Aware Caches," *Proc. Int'l Conf. Design, Automation and Test in Europe,* 2007.

[26] R. Pasko et al., "A New Algorithm for Elimination of Common Subexpressions," *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems,* vol. 18, no. 1, pp. 58-68, Jan. 1999.

[27] N. Quach, "High Availability and Reliability in the Itanium Processor," *IEEE Micro,* vol. 20, no. 5, pp. 61-69, Sept./Oct. 2000.

[28] P. Hazucha and C. Svensson, "Impact of CMOS Technology Scaling on the Atmospheric Neutron Soft Error Rate," *IEEE Trans. Nuclear Science,* vol. 47, no. 6, pp. 2586-2594, Dec. 2000.

[29] J. Keane et al., "Method for Qcrit Measurement in Bulk CMOS Using a Switched Capacitor Circuit," *Proc. VLSI Symp.,* 2007.

[30] S. Lin and D. Costello, *Error Control Coding,* second ed. Prentice Hall, 2004.

[31] I.S. Reed and M.T. Shih, "VLSI Design of Inverse-free Berlekamp-Massey Algorithm," *Computers and Digital Techniques, IEE Proc.,* vol. 138, no. 5, pp. 295-298, 1991.

[32] F. Sun, K. Rose, T. Zhang, "On the Use of Strong bch Codes for Improving Multilevel Nand Flash Memory Storage Capacity," *Proc. IEEE Workshop Signal Processing Systems,* 2006.

[33] B. Alorda, G. Torrens, S. Bota, and J. Segura, "Static and Dynamic Stability Improvement Strategies for 6T CMOS Low-power SRAMs," *Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE),* 2010.

Somnath Paul (S'07) received the BE degree in electronics and telecommunication engineering from Jadavpur University, Kolkata, India, in 2005. He is currently working toward the PhD degree in computer engineering at Case Western Reserve University, Cleveland, Ohio. He was a design engineer with Advanced Micro Devices, Bengaluru, India. He has also held internship positions at Intel and Qualcomm. His research interests include development of novel hardware frameworks for reconfigurable architectures and hardware/software codesign for yield improvement in nanoscale technologies. He is a student member of the IEEE.

Fang Cai (S'10) received the BS degree in electrical engineering from Zhejiang University, Hangzhou, China, in 2007. He is currently working toward the PhD degree in electrical engineering and computer science at Case Western Reserve University, Cleveland, Ohio. His current research interests include the design of VLSI architectures for communications and digital signal processing, with the emphasis on error-correcting coding, especially on iterative channel coding. He is a student member of the IEEE.

Xinmiao Zhang (S'04-M'05) received the BS and MS degrees in electrical engineering from Tianjin University, China, in 1997 and 2000, respectively, and the PhD degree in electrical engineering from the University of Minnesota-Twin Cities, in 2005. Since then, she has been with Case Western Reserve University, where she is currently a Timothy E. and Allison L. Schroeder associate professor with the Department of Electrical Engineering and Computer Science. Her research interests include VLSI architecture design for communications, cryptosystems, and digital signal processing. She was a recipient of the US National Science Foundation (NSF) CAREER Award in 2009, and the Best Paper Award at the ACM Great Lakes Symposium on VLSI 2004. She is the coeditor of the book *Wireless Security and Cryptography: Specifications and Implementations* (CRC Press, 2007) and the guest editor for the Springer MONET Journal Special Issue on *Next Generation Hardware Architectures for Secure Mobile Computing*. She is a member of the Circuits and Systems for Communications and VLSI Systems and Applications technical committees of the IEEE Circuits and Systems Society and the Design and Implementation of Signal Processing Systems technical committee of the IEEE Signal Processing Society. She has served on technical program committees of ACM Great Lakes Symposium on VLSI, IEEE Workshops on Signal Processing Systems, IEEE Global Communications Conference, and the reviewer committees of IEEE International Symposium on Circuits and Systems. She is currently an associate editor for the *IEEE Transactions on Circuits and Systems-I: Regular Papers*. She is a member of the IEEE.

Swarup Bhunia (S'00-M'05-SM'09) received the BE (Hons) degree from Jadavpur University, Kolkata, India, the MTech degree from the Indian Institute of Technology (IIT), Kharagpur, and the PhD degree from Purdue University, Indiana, in 2005. Currently, he is an assistant professor of electrical engineering and computer science at Case Western Reserve University, Cleveland, Ohio. He has more than 10 years of research and development experience with over 100 publications in peer-reviewed journals and premier conferences in the area of VLSI design, CAD, and test techniques. His research interests include low-power and robust design, hardware security and protection, adaptive nanocomputing, and novel test methodologies. He has worked in the semiconductor industry on RTL synthesis, verification, and low-power design for about three years. He received Semiconductor Research Corporation (SRC) Technical Excellence Award (2005), Best Paper Award in International Conference on Computer Design (ICCD 2004) and Latin American Test Workshop (LATW 2003), best paper nomination in Asia and South Pacific Design Automation Conference (ASP-DAC 2006) and Hardware Oriented Test and Security (HOST 2010), nomination for John S. Diekhoff Award at Case Western Reserve University (2010), and SRC Inventor Recognition Award (2009). He has served as the guest editor of the*IEEE Design and Test of Computers* (2010), in the editorial board of the *Journal of Low Power Electronics* (JOLPE) and in the technical program committee of number of major IEEE/ACM conferences. He is a senior member of the IEEE.