Leveraging Access Locality for the Efficient Use of Multibit Error-Correcting Codes in L2 Cache

Hongbin Sun, Nanning Zheng, *Fellow*, *IEEE*, and Tong Zhang, *Senior Member*, *IEEE*

Abstract—It is almost evident that SRAM-based cache memories will be subject to a significant degree of parametric random defects if one wants to leverage the technology scaling to its full extent. Although strong multibit error-correcting codes (ECC) appear to be a natural choice to handle a large number of random defects, investigation of their applications in cache remains largely missing arguably because it is commonly believed that multibit ECC may incur prohibitive performance degradation and silicon/energy cost. By developing a cost-effective L2 cache architecture using multibit ECC, this paper attempts to show that, with appropriate cache architecture design, this common belief may not necessarily hold true for L2 cache. The basic idea is to supplement a conventional L2 cache core with several special-purpose small caches/buffers, which can greatly reduce the silicon cost and minimize the probability of explicitly executing multibit ECC decoding on the cache read critical path, and meanwhile, maintain soft error tolerance. Experiments show that, at the random defect density of 0.5 percent, this design approach can maintain almost the same instruction per cycle (IPC) performance over a wide spectrum of benchmarks compared with ideal defect-free L2 cache, while only incurring less than 3 percent of silicon area overhead and 36 percent power consumption overhead.

Index Terms—Cache, defect tolerant, error-correcting code.

1 Introduction

Continuous CMOS technology scaling makes the design of robust and high-density SRAM-based cache an increasingly challenging task [1]. Potential faults in SRAM can be parametric/catastrophic defects or transient soft errors, both of which are becoming increasingly serious as the technology feature size shrinks. In conventional design practice, memory defects are handled by using spare (or redundant) rows, columns, and/or words to repair (i.e., replace) the defective ones, while soft errors are compensated by error-correcting codes (ECC) such as *single-error-correcting and double-error-detecting* (SEC-DED) codes that are being widely used in L2 cache of modern microprocessors [2], [3]. As the technology continues to scale down, the increasingly severe process variability tends to render future SRAM subject to a parametric random defect of 0.1 percent or even higher [4]. As a result, traditional repair-only defect tolerance strategy may no longer be sufficient to ensure high enough yield, which has motivated recent work on extending the role of ECC for compensating both soft errors and defects in cache memories [5], [6]. In [5], the authors developed techniques that allow the use of the existing SEC-DED codes to handle defects for the cache blocks consisting of a single defect while maintaining soft error tolerance at the cost of memory communication bandwidth loss, and hence, noticeable instructions per cycle (IPC) degradation. In [6], 2D array codes (or product codes) [7] are used to handle clustered soft errors and/or defects. Nevertheless, since one 2D array codeword protects many cache blocks altogether, the use of array codes may incur significant energy cost and IPC degradation in the presence of a large amount of random defects.

This work concerns the feasibility and potential of using multibit ECC (i.e., the ECC that can correct multiple errors within one codeword) in each individual cache data with small size (e.g., 32- or 64-bit) to tolerate a large amount of random defects in L2 cache without the loss of soft error tolerance. In fact, the use of multibit ECC for memory defect tolerance has been well studied, two decades ago [8], [9], where researchers developed several multibit ECC that can realize a shorter decoding latency than conventional multibit ECC such as the BCH code at the cost of coding redundancy. Therefore, this work does not intend to develop any new multibit ECC; instead, we focus on how to enable the effective use of multibit ECC in L2 cache. The use of multibit ECC for cache memories clearly faces two problems: 1) The decoding latency of multibit ECC is (much) longer than that of the existing SEC-DED code. Because cache memories are extremely read-latency sensitive in nature, the longer decoding latency appears as the most critical issue. 2) Compared with the existing SEC-DED code, multibit ECC tends to incur much higher coding redundancy, particularly when being used to protect relatively small cache data (e.g., 32- or 64-bit). It is arguably true that the above two issues make many researchers tend to easily preclude the possibility of using multibit ECC in cache memories.

Although it is most likely true that multibit ECC can be hardly used in L1 cache, we contend that, contrary to the common belief, multibit ECC can indeed hold a great promise to play a key role in L2 cache design as the technology scaling begins to incur a significant degree of parametric random SRAM cell defects. First, let us revisit the above quoted two problems regarding the use of multibit ECC in cache memories. Since cache blocks consisting of one

- *H. Sun and N. Zheng are with the Xi'an Jiaotong University, No. 28, West Xianning Road, Xi'an, Shaanxi 710049, P.R. China. E-mail: {sunsir, nnzheng}@mail.xjtu.edu.cn.*
- *T. Zhang is with the Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180. E-mail: tzhang@ecse.rpi.edu.*

Manuscript received 28 June 2008; revised 1 Feb. 2009; accepted 23 Feb. 2009; published online 17 Mar. 2009.
Recommended for acceptance by C. Bolchini.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-06-0322.
Digital Object Identifier no. 10.1109/TC.2009.45.

or more defective cells can be identified during memory testing [8], it is very intuitive that a better choice is to apply multibit ECC to the cache blocks only whenever necessary instead of uniformly protecting all the cache blocks using multibit ECC. Such a simple selective use of multibit ECC may largely alleviate the above two quoted problems, i.e., 1) since not all cache read operations invoke multibit ECC decoding, the impact on the overall cache performance may be reduced and 2) the amount of memory cells required to store the coding redundancy can be accordingly reduced. Intuitively, implementation of the selective use of multibit ECC must perform runtime table lookup to check whether or not the cache block being accessed should be protected by the multibit ECC. This can be realized by content-addressable memory (CAM) [10], e.g., the authors of [11] proposed to use CAM to store the address of the memory blocks in which the defects are handled by ECC. However, although a direct realization of selective use of multibit ECC accompanied by CAM is quite straightforward, its effectiveness may be inadequate in the presence of a relatively high random defect density for two main reasons: 1) as the random defect density increases, a larger percentage of cache read operations may invoke multibit ECC decoding, which will directly degrade the overall system performance such as IPC and 2) since the energy consumption of CAM is substantially larger than that of normal SRAM and the size of CAM will increase as the random defect density increases, a significant energy consumption overhead will be incurred.

Following the same simple design principle (i.e., selective use of multibit ECC accompanied by CAM), this work develops methods that can reduce the cache memory performance degradation and energy consumption overhead in the presence of a large amount of random defects. The basic ideas are simple and intuitive, and are briefly described as follows. To reduce the cache memory performance degradation and energy consumption, we propose to add a small buffer that keeps the copies of the most recently accessed cache blocks being protected by multibit ECC. Because of the well-known cache access temporal locality, the use of this small duplication buffer can largely reduce the occurrence of explicit multibit ECC decoding. To further reduce the energy consumption incurred by CAM access, we add another small CAM to store the addresses of the most recently accessed cache blocks that are not protected by multibit ECC. Again, because of the cache access temporal locality, it may avoid much of explicit access to the bigger CAM that supports the selective use of multibit ECC. It should be pointed out that, if all of the error-correcting capabilities of an ECC are devoted to defect tolerance for one cache block in L2 cache, this cache block will be left unprotected for soft errors. For *clean* cache blocks, an extra error-detecting capability would be sufficient since correct data can be fetched from the lower level memory. Data integrity problem happens only if soft errors occur in *dirty* cache blocks that are not backed up elsewhere. A straightforward solution is to always write such dirty L2 cache blocks back to the next level memory, which may incur intermemory-hierarchy bandwidth loss. In this work, we propose to add a small buffer inside L2 cache to keep the copies of the most recent write-back data from L1 cache, which can largely reduce the frequency of the write back from L2 cache to the next level memory. We note that, in case of multithreading applications, the technique proposed in [12] can also be used to strengthen soft error tolerance.

The effectiveness of the proposed methods has been demonstrated in a case study that assumes a random defect density of 0.5 percent and the use of *double-error-correcting and triple-error-detecting* (DEC-TED) code based on BCH code as the multibit ECC. The SimpleScalar and Cacti tools are used for processor simulation and cache memory modeling at 45-nm technology node. Results show that, under the defect density of 0.5 percent, the developed design approach can maintain almost the same IPC performance over a wide spectrum of SPEC2000 benchmarks compared with ideal defect-free L2 cache, while only incurring less than 3 percent of silicon area overhead and 36 percent of energy consumption overhead.

The rest of this paper is organized as follows: Section 2 presents motivating examples to further justify this work. Section 3 describes the proposed L2 cache architecture that can effectively handle a large amount of random defects. Sections 4 and 5 present the experimental methodology and results to demonstrate the effectiveness of the proposed design solution. The conclusions are drawn in Section 6.

2 Motivation

In modern microprocessors, each L2 cache block is typically partitioned into several equal-sized subblocks and each subblock is protected with an SEC-DED code for soft error tolerance. In this work, we categorize cache subblocks based on the number of defective cells they contain:

- A subblock that does not have any defective cell is called a good subblock, which is denoted as *g-subblock*.
- A subblock that has one defective cell is called a single-defect subblock, which is denoted as *s-subblock*.
- A subblock that has two or more defective cells is called a multidefect subblock, which is denoted as *m-subblock*.

Furthermore, a cache block that only contains g-subblocks is called a *g-block*, a cache block that contains both g-subblocks and s-subblocks is called an *s-block*, and a cache block that contains one or more m-subblocks is called an *m-block*. Let P_{g-sblk}, P_{s-sblk}, and P_{m-sblk} represent the probabilities that an n-bit cache subblock is a g-subblock, s-subblock, and m-subblock, respectively, which can be calculated as follows given the random cell defect density λ

$$\begin{aligned} P_{g-sblk} &= (1-\lambda)^n, \\ P_{s-sblk} &= n \times (1-\lambda)^{n-1} \times \lambda, \\ P_{m-sblk} &= 1 - P_{g-sblk} - P_{s-sblk}. \end{aligned}$$

Furthermore, let P_{g-blk}, P_{s-blk}, and P_{m-blk} represent the probabilities that a block consisting of k subblocks is a g-block, s-block, and m-block, respectively, which can be calculated as

$$\begin{aligned} P_{g-blk} &= P_{g-sblk}^k, \\ P_{s-blk} &= (1 - P_{m-sblk})^k - P_{g-blk}, \\ P_{m-blk} &= 1 - P_{g-blk} - P_{s-blk}. \end{aligned}$$

For example, following the same L2 cache configuration in AMD Opteron processor [3], let us consider a 1 MB L2 cache in which each cache block contains eight subblocks and each cache subblock contains 64 bits. Accordingly, we can

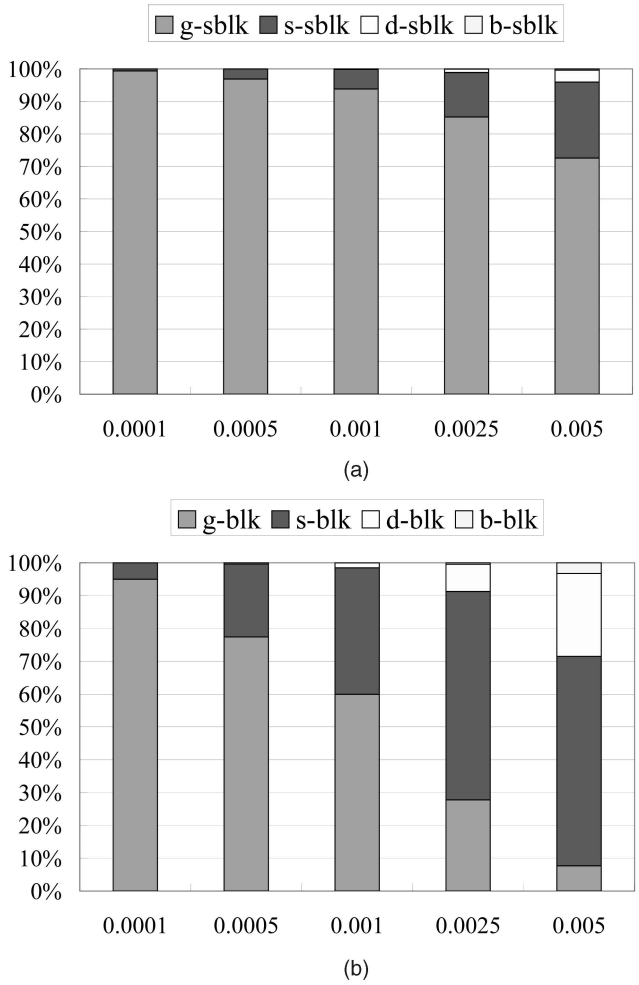


Fig. 1. Histogram of different types of (a) cache subblocks and (b) cache blocks under different random defect rates.

calculate those different probabilities under different random cell defect densities, as shown in Fig. 1.

As random defect density increases, the percentages of m-subblocks and m-blocks quickly become nonnegligible, e.g., under the defect density of 0.5 percent, the possibilities that a cache block is a g-block, s-block, and m-block are 7.7 percent, 63.8 percent, and 28.5 percent, respectively. Clearly, if we use redundancies to repair all the cell defects, 92.3 percent of all cache blocks should be repaired. Even though we use redundancies only to repair all the m-blocks and use SEC-DED code to handle the s-blocks, the redundancy overhead is still significant (i.e., 28.5 percent of all cache blocks should be repaired). Although recently proposed CAM-based self-repair approaches [13] may achieve better memory defect tolerance efficiency than traditional repair approaches, they tend to become infeasible under high defect density, briefly explained as follows. Those techniques demand a CAM with 41,944 entries to record all the addresses of defective cells in the 1 MB L2 cache under the defect density of 0.5 percent. According to the memory modeling tool Cacti 5 [14], the area overhead can be as high as 10.3 percent. More importantly, since the CAM is searched for every memory access when using self-repair approach, the dynamic energy consumption of the CAM is 95 times larger than L2 cache itself, which makes the self-repair approaches infeasible for L2 cache defect tolerance at the relatively high defect density. The above discussion clearly suggests that state-of-the-art defect tolerance strategies may no longer be adequate under a relatively high random defect density, and we believe that extending the role of ECC for both soft error tolerance and defect tolerance appears almost inevitable. It should be emphasized here that ECC are used to complement with the existing redundant repair, other than replacing it, for the purpose of memory defect tolerance.

TABLE 1
Coding Redundancy Comparison between SEC-DED and DEC-TED Codes

Data bits	32	64	128
SEC-DED check bits	7	8	9
DEC-TED check bits	12	14	16

The above simple calculations indicate that a stronger multibit ECC may be necessary in order to enable a dual-role of ECC. However, a straightforward use of multibit ECC in L2 cache, i.e., all the L2 cache blocks are uniformly protected by the same multibit ECC, will incur prohibitive cache access latency and silicon area penalty. For example, compared with SEC-DED codes, BCH-based DEC-TED codes that can correct up to two errors and detect the occurrence of three errors demand much more check bits as listed in Table 1. More importantly, the decoding latency of BCH-based DEC-TED codes can be much longer than that of SEC-DED codes. The decoding of SEC-DED codes only involves simple XOR operations and can be efficiently parallelized while the decoding of BCH-based DEC-TED codes involves more complex Galois Field arithmetics and even its fully parallel realization, at the cost of a significant silicon area overhead, still suffers from relatively long latency [15]. Although there exist other types of DEC-TED codes with less decoding latency [8], [9], they will incur much more check bits than BCH-based DEC-TED codes and their decoding latencies are still considerably longer than that of SEC-DED codes. As pointed out earlier, we should selectively use multi-ECC for cache blocks whenever necessary, and a straightforward realization of such selective use of multi-ECC tends to incur significant access latency, silicon area, and energy consumption overhead.

3 Proposed L2 Cache Architecture

In this section, we present an L2 cache architecture that enables the selective use of strong multibit ECC to strengthen defect tolerance at small cache access latency, silicon area, and energy consumption overhead, while maintaining soft error tolerance. Fig. 2 shows the overview of this proposed L2 cache architecture, which consists of three main blocks:

1. A conventional L2 cache core that follows the current L2 cache design practice (e.g., see [3]) and uniformly protects all the subblocks using SEC-DED codes.
2. A multibit ECC core that contains a fully associative multibit ECC cache (denoted as M-ECC cache), the corresponding ECC encoder/decoder, and two small accessory buffers. The fully associative M-ECC cache is tagged by the location of each m-subblock and stores the corresponding multi-ECC check bits. As

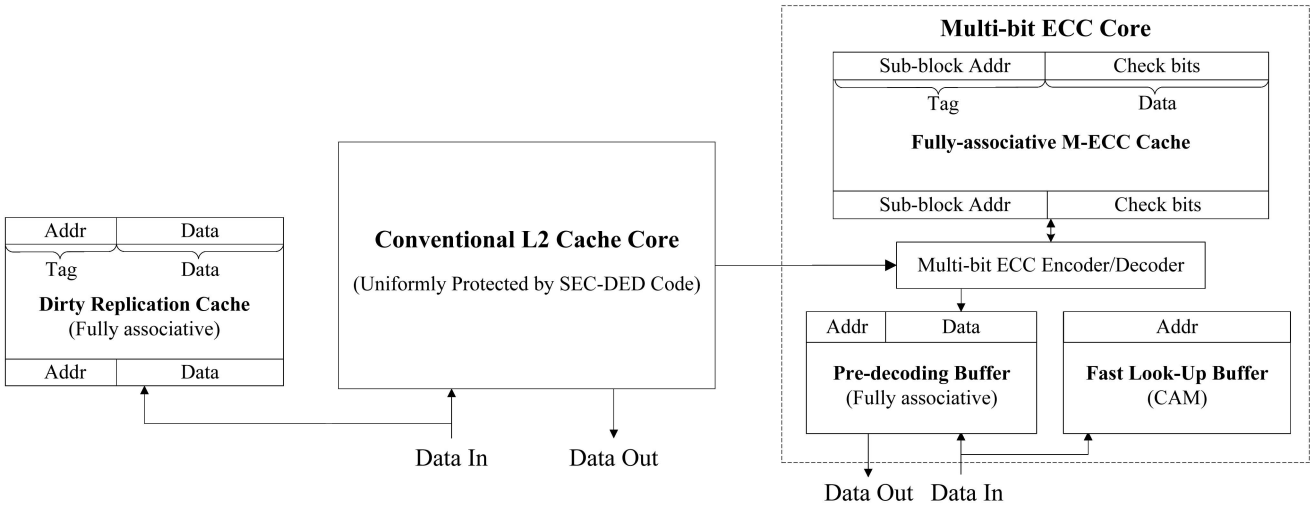


Fig. 2. Overview of proposed L2 cache architecture.

explained later, the two small buffers, i.e., the *predecoding buffer* and *fast lookup (FLU) buffer*, as shown in Fig. 2, are used to reduce the probabilities that the M-ECC cache is accessed and multibit ECC decoding is explicitly invoked.

3. A small fully associative dirty replication cache that is used to keep copies of most recent write-back data from L1 cache, which is used to ensure soft error tolerance as described later.

3.1 Multibit ECC Core for Strong Defect Tolerance

As illustrated in Fig. 2, the fully associative M-ECC cache holds the location and the corresponding multibit ECC check bits of each m-subblock. Since the M-ECC cache may not always be full, we can add one valid bit to each tag to indicate whether it indeed contains the check bits for one m-subblock, which can be leveraged to reduce the M-ECC cache access energy. The tag portion of this M-ECC cache can be supplied from embedded/external nonvolatile memories such as flash memory. If a built-in self-test (BIST) unit [16], [17] is available, this information can be periodically updated to handle the defects that develop in the field. The operation of this fully associative M-ECC cache itself is very simple, i.e., in case of write, the m-subblock data are encoded by the multibit ECC encoder and the check bits are stored in the corresponding entry in the M-ECC cache; in case of read, the check bits are fetched from the M-ECC cache and sent to the multibit ECC decoder.

As pointed out earlier, if we only use this fully associative M-ECC cache to realize the selective multibit ECC protection in L2 cache, it may result in prohibitive cache access latency and energy consumption cost as the random defect density increases. To address these two issues, as illustrated in Fig. 2, we propose to add a small fully associative cache, denoted as predecoding buffer, that keeps the copies of most recently accessed m-blocks. This fully associative predecoding buffer employs the least recently used (LRU) policy for replacement when it is full. In case of an L2 cache read hit, this predecoding buffer is searched before accessing the M-ECC cache, and if a hit occurs, the cache block stored in this buffer will be sent out without incurring explicit M-ECC cache access and multibit ECC decoding. Because of the typically high degree of cache access temporal locality, the use of this small predecoding buffer can obviate a large percentage of multibit ECC decoding operations and a certain degree of M-ECC cache access. As a result, the average cache access latency and energy consumption, incurred by explicit M-ECC cache access and multibit ECC decoding, can be reduced.

Although the predecoding buffer can reduce the occurrence of M-ECC cache access in case that an m-block is being accessed, M-ECC cache will be explicitly accessed if a g-block or an s-block (i.e., those cache blocks that are not protected by the multibit ECC) is being accessed. Since most cache blocks are either g-blocks or s-blocks, the M-ECC cache may still be accessed frequently, which can make the energy consumption incurred by accessing the fully associative M-ECC cache tend to be high. To further reduce such energy consumption cost, we propose to add another small CAM, called FLU buffer, to keep the address of recently accessed cache blocks that are not protected by the multibit ECC. If an FLU hit occurs, which means the cache block being accessed is either a g-block or an s-block, then it is not necessary to check the M-ECC cache. The FLU buffer is dynamically updated using LRU policy similar to the predecoding buffer.

By supplementing the main M-ECC cache with the small predecoding and FLU buffers, we can largely reduce the impact of M-ECC cache access and multibit ECC decoding on the overall L2 cache access latency and energy consumption, which will be further demonstrated by experimental results presented later. Fig. 3 shows the overall operation flow of the M-ECC core described above. Finally, we note that, because the storage capacity of the entire M-ECC core can be much less than that of the L2 cache core, it is reasonable to expect that the SRAM cell defects in M-ECC core can be solely handled by conventional redundancy-repair strategy at modest silicon cost.

3.2 Dirty Replication Cache for Soft Error Tolerance

This section addresses how to maintain soft error tolerance while using the existing error-correcting capability for defect tolerance. For cache hierarchy using write-back policy, when all the error-correcting capability of ECC is devoted to realizing defect tolerance, the defective cache subblock clearly becomes vulnerable to soft errors. If a subblock does not contain any defective cell (i.e., it is a g-subblock), the

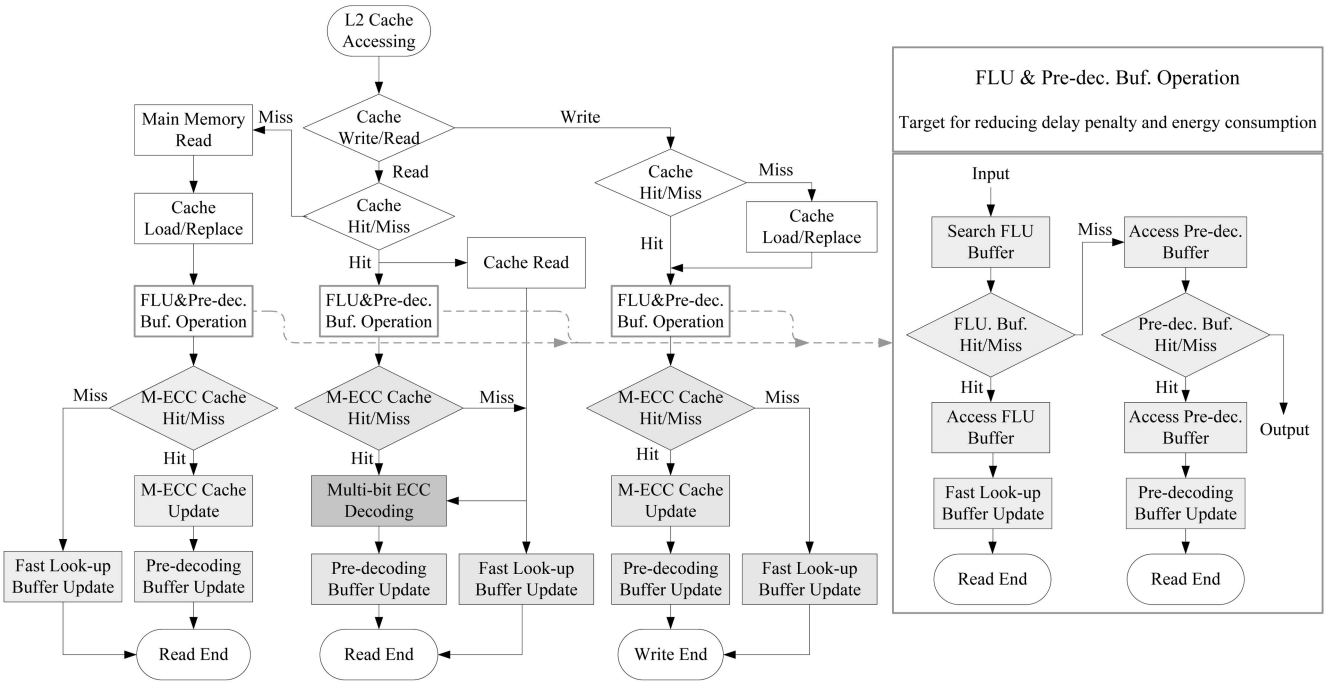


Fig. 3. Operation flow chart of the M-ECC core.

uniformly applied SEC-DED code is sufficient to ensure soft error tolerance as in conventional design practice. However, the occurrence of soft errors may only be detectable but cannot be corrected for

- an s-subblock that contains a single defect and is protected by SEC-DED code, or
- an m-subblock that contains more than one defect and is protected by multibit ECC.

Hence, we must restore the full soft error tolerance for those cache blocks. In this regard, we note that not all the soft errors demand explicit error correction by ECC. If the cache block corrupted by soft errors is clean, then error detection by ECC is sufficient since the correct data can be obtained from the lower level memory. Data integrity problem occurs only if the corrupted cache blocks are dirty. Therefore, in the following, we only focus on the realization of soft error tolerance for those dirty cache blocks.

One possible solution is to simply use write-through policy instead of write-back policy (i.e., every write to the L2 cache causes a synchronous write to its lower level memory, for example, main memory). In general, write-through policy tends to incur very significant bandwidth overhead, leading to noticeable degradation of the overall computing system performance. As a result, write-through policy is less popular than its write-back counterpart in practice. Therefore, this work is only interested in write-back cache. In the context of write-back cache, it is intuitive that the problem can be solved if we are allowed to selectively carry out write-through for dirty defective cache blocks, referred to as *assurance update* [5]. However, under high random defect densities, even assurance update may severely increase the intercache-hierarchy communication burden and result in performance degradation. The authors of [5] proposed to maintain the dirtiness and assurance update at the cache subblock level, which can reduce the communication burden compared with keeping assurance update at the cache block level. However, the incurred implementation overhead and performance degradation tend to be nonnegligible under relatively high random defect density (e.g., 0.1 percent and higher).

In this work, following the same principle of selective L2 cache write-through, we propose to directly add a small dirty replication cache (denoted as DR cache) within the L2 cache itself. The small embedded DR cache simply keeps the copies of most recent write-back cache blocks (i.e., dirty cache blocks) from L1 cache to ensure soft error tolerance for those dirty cache blocks. In other words, the soft error tolerance is realized by data duplication other than explicit error correction within L2 cache. The small DR cache is fully associative and its operation flow is shown in Fig. 4: During

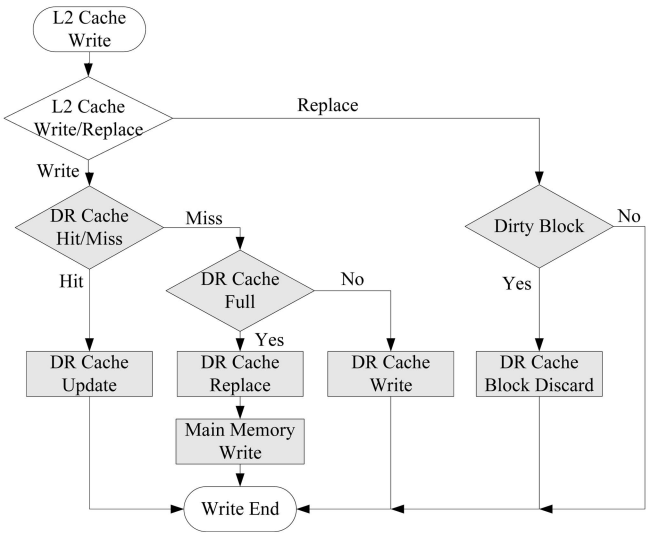


Fig. 4. Operation flow chart of the dirty replication (DR) cache.

TABLE 2
Default Configuration Parameters Used in Simplescalar

Configuration Parameters	Value
Processor	
Functional Units	4 integer ALUs 1 integer multiplier/divider 4 FP ALUs 1 FP multiplier/divider
LSQ Size	8 Instructions
RUU Size	16 Instructions
Fetch Width	4 Instructions/cycle
Decode Width	4 Instructions/cycle
Issue Width	4 Instructions/cycle
Commit Width	4 Instructions/cycle
Fetch Queue Size	4 Instructions
Cycle Time	1 ns
Cache and Memory Hierarchy	
L1 Instruction Cache	64KB, 2-way, 64-byte blocks 1 cycle latency
L1 Data Cache	64KB, 2-way, 64-byte blocks 1 cycle latency
L2	1MB unified, 8-way 64-byte blocks 13 cycle latency
Main Memory	300 cycle latency
Branch Logic	
Predictor	combined, bimodal 2KB table two-level 1KB table 8 bit history
BTB	512 entry, 4-way
Miss-prediction Penalty	3 cycles

L2 cache write, in case of a DR cache hit, the corresponding cache block in the DR cache will be updated together with the cache block in L2 cache itself; in case of a DR cache miss, if the DR cache is not full, the incoming data will be added into it as a new entry, otherwise the copy of the least recently used cache block in the DR cache will be replaced and written back to the next level memory. Moreover, if a dirty L2 cache block is replaced, its copy in the DR cache will be discarded simultaneously. Equipped with this small DR cache, the L2 cache can ensure that there is always a backup for each cache block in either the DR cache or next level memory, no matter the cache block is clean or dirty. Therefore, whenever a soft error makes a cache block uncorrectable, a backup can always be found, leading to a full restoration of soft error tolerance in L2 cache. As shown by our experiments described later, with a modest size of DR cache, most write back from L1 cache can result in a hit of this small cache, leading to almost no impact on intercache-hierarchy communication load.

Finally, it should be pointed out that, in the above proposed cache architecture, all the extra memory modules are essentially based on CAM, and hence, their soft error tolerance may not be trivial. Researchers recently presented several techniques that can correct or detect soft errors in CAM, e.g., see [18], [19], [20]. Another possibility is to simply upscale the transistors in those small memory modules to strengthen the soft error immunity.

4 Evaluation Methodology

4.1 Platform and Benchmark

To evaluate the performance of the above presented L2 cache architecture, we carry out simulations using the popular *SimpleScalar 3.0* simulator [21]. Table 2 lists the simulator configuration parameters. The cache memory defect density is set as 0.5 percent. We assume the use of BCH-based DEC-TED code, each cache subblock contains 64 bits, and cache subblocks containing more than two errors are repaired by redundancy. We assume that the BCH DEC-TED decoder has a decoding parallelism of 2 and uses the PGZ decoding algorithm [22], leading to an estimated decoding latency of 82 cycles. We use Cacti 5, the latest version of a widely used cache modeling tool Cacti [14], to estimate the characteristics of the main L2 cache core, M-ECC cache, predecoding buffer, FLU buffer, and DR cache at 45 nm node. Regarding the extra logic circuit overhead, using verilog HDL synthesized with TSMC 65 nm standard cell library, we designed the circuits to implement the entire operation flows including ECC codec. After being scaled down to 45 nm, the overall area is less than $0.01\ \mathrm{mm}^2$, i.e., the area overhead of the operation flow is about 0.2 percent of the area of the L2 cache core. Hence, the following evaluation only focuses on the overhead of additional memories in the proposed architecture.

TABLE 3
Benchmarks Used for Performance Evaluation

SPEC2000	Phase FFWD	Phase RUN	L1 misses / Million ins.	L2 hits / Million ins.	IPC
164.gzip	1.0B	400M	2875	4468	1.7520
175.vpr	100M	400M	1343	1469	1.6048
176.gcc	1.0B	400M	48618	75468	1.3526
177.mesa	500M	400M	6329	6585	1.5300
179.art	1.2B	300M	35055	24021	0.1872
181.mcf	200M	300M	9461	7684	0.4504
183.equake	100M	400M	3816	3795	1.7033
188.ammp	100M	300M	16825	983	0.1830
197.parser	100M	400M	1962	2652	1.2910
255.vortex	1.0B	400M	19409	20115	1.1866
256.bzip2	200M	400M	4585	5326	1.2925

Table 3 shows the 12 benchmarks used in our experiment, including 7 SPEC2000 integer benchmarks and 5 SPEC2000 floating-point benchmarks [23]. For each benchmark, we searched and simulated over the instruction sequences with relatively high L2 access rate. Table 3 lists the number of instructions skipped to reach the phase start, i.e., the fast-forward (FFWD) column, and the number of instructions simulated, i.e., the RUN column. Table 3 also shows the average number of L1 cache misses and L2 cache hits per million instructions and the corresponding average IPC.

4.2 Target Systems

Since the M-ECC core for defect tolerance and the DR cache for soft error tolerance are completely independent from each other, we can evaluate them separately. Regarding M-ECC core for defect tolerance, we consider the following four different scenarios:

- Base: The L2 cache is defect-free; hence, does not employ any defect tolerant functions.
- M-ECC-only: The L2 cache core is only equipped with an M-ECC cache, so each L2 cache access incurs one M-ECC cache access.
- M-ECC-pbuf: The L2 cache core is equipped with an M-ECC cache and a predecoding buffer.

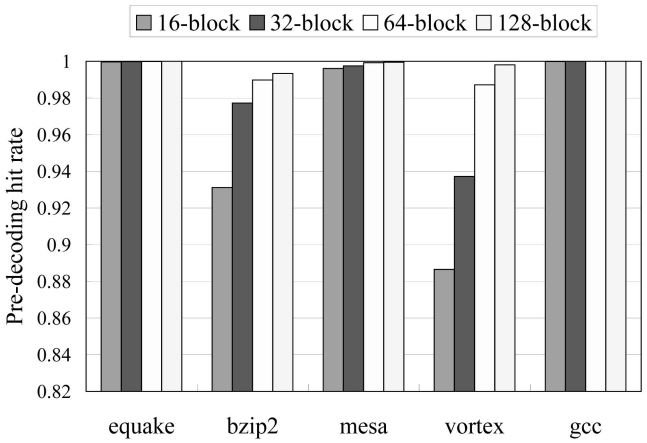


Fig. 5. Multibit ECC predecoding hit rates for Predecoding buffers of 16, 32, 64, and 128 blocks. The L2 cache is fixed to be 1 MB, 8-way associative (only the five worst case benchmarks are shown).

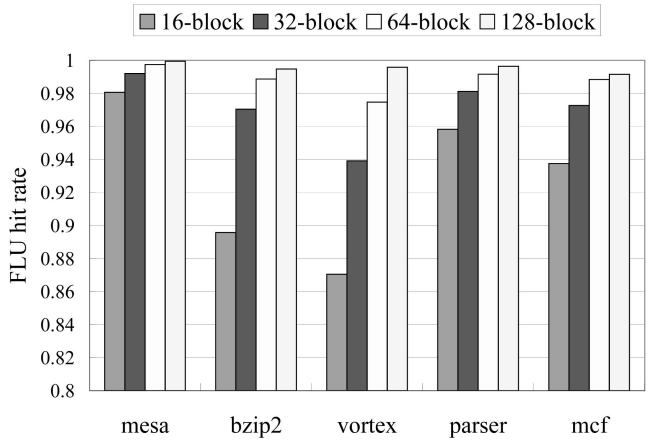


Fig. 7. FLU hit rate for FLU buffers of 16, 32, 64, and 128 entries. The L2 cache is fixed to be 1 MB, 8-way associative (only the five worst case benchmarks are shown).

- M-ECC-pfbuf: The L2 cache core is equipped with the entire M-ECC core consisting of the M-ECC cache, predecoding buffer, and FLU buffer.

We evaluate the performance and overhead of the dirty replication cache, which is further compared with both the block-level assurance update, denoted as AU-blk, and the subblock-level assurance update, denoted as AU-sblk [5].

5 Evaluation Results

5.1 M-ECC Core

5.1.1 Size of the Predecoding Buffer

We first investigate how the size of the fully associative predecoding buffer affects the cache performance. By setting L2 cache core to be 1 MB, 8-way associative as shown in Table 2, we vary the size of the predecoding buffer from 16 cache blocks up to 128 cache blocks. Fig. 5 shows the predecoding hit rate for five different benchmarks. Clearly, the predecoding hit rate increases with the increase of the predecoding buffer size. With the size of 64 cache blocks and higher, the hit rate can be larger than 98.4 percent, on average, which means that most explicit multibit ECC decoding can be avoided. Furthermore, for a 64-block predecoding buffer, we conduct experiments to examine the predecoding buffer hit rate by varying the number of associativity of the L2 cache. Fig. 6 shows the predecoding buffer hit rates with the number of L2 cache associativity varying from 4-way to 16-way. As the set associativity increases, the predecoding buffer hit rate decreases. Nevertheless, even for the 16-way associative L2 cache, a 64-block predecoding buffer can still achieve a hit rate as high as 98 percent. The above results suggest that 64-block predecoding buffer appears to be an appropriate choice in this experimental setup.

5.1.2 Size of the FLU Buffer

The proposed cache design leverages the FLU buffer to reduce the energy consumed by searching the M-ECC cache. In the following experiment, we intend to determine the appropriate size for the fully associative FLU buffer. Fig. 7 shows the FLU hit rates with different buffer sizes varying from 16 entries to 128 entries. For FLU buffers with 64 or more entries, the FLU hit rates are larger than 97.5 percent, which means most g-block and s-block accesses do not need to search the M-ECC cache to check their classification. For a 64-entry FLU buffer, we conduct experiments to examine

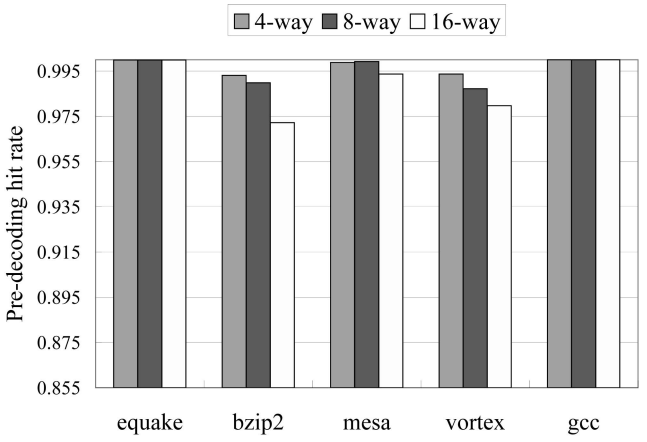


Fig. 6. Multibit ECC predecoding hit rate for 4-way, 8-way, and 16-way set associative 1 MB L2 cache. The Predecoding buffer is fully associative with 64 blocks (only the five worst case benchmarks are shown).

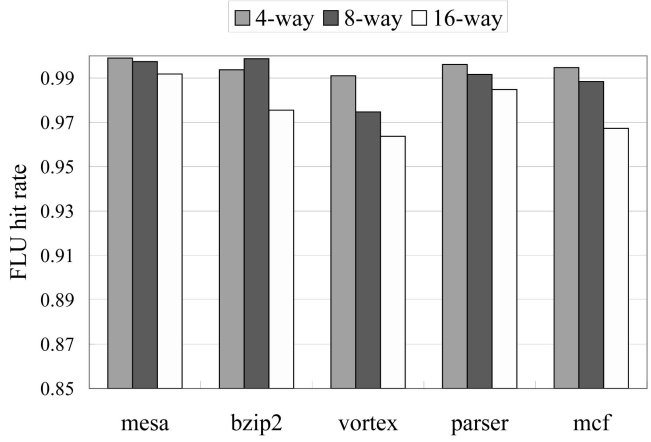


Fig. 8. FLU hit rate for 4-way, 8-way, and 16-way set associative 1 MB L2 cache. The FLU buffer is fully associative with 64 entries (only the five worst case benchmarks are shown).

TABLE 4
Cacti Report for L1 Cache, L2 Cache, M-ECC Cache, DR Cache, Predecoding Buffer, and FLU Buffer

Memory	Size	Associativity	Block size	Access time (ns)	Cycle time (ns)	Dynamic Energy (nJ)	Area (mm^2)
L1 instruction/data cache	64KB	2	64	0.53	0.12	0.0100	0.3216
L2 cache	1MB	8	64	1.60	0.18	0.0379	4.3483
M-ECC cache	16KB	full	2	0.39	0.05	0.6820	0.0840
Pre-dec. buffer / DR cache	4KB	full	64	0.31	0.05	0.0080	0.0225
FLU buffer (64 entries)	-	full	-	0.25	0.02	0.0072	0.0009

the FLU buffer hit rate by varying the number of associativity of the L2 cache. Fig. 8 shows the FLU buffer hit rate with the number of L2 cache associativity varying from 4-way to 16-way. Even in the worst case (i.e., 16-way), it can still maintain a high hit rate above 96 percent. Considering the trade-off between performance and cost, we may choose 64-entry as the proper size of the FLU buffer.

5.1.3 Overall Performance Comparison

To evaluate the latency, power, and area overhead of the proposed design, we set the size of the M-ECC cache and predecoding buffer as 16 KB and 4 KB (i.e., 64 blocks), respectively, and the FLU buffer has 64 entries. Table 4 lists the corresponding modeling results produced by Cacti at 45 nm technology node, including access/cycle time, dynamic energy, and area. Clearly, the access time of the M-ECC cache, predecoding buffer and FLU buffer is much less than that of the L2 cache, and can even be comparable to the cycle time of the L2 cache. Hence, it is reasonable to assume that access to the M-ECC cache can be parallel with the access to the L2 cache core and will not add any extra cycles. However, the latency caused by the multibit ECC decoding can significantly impact the IPC performance. According to the normalized IPC comparison among three target systems shown in Fig. 9 (since M-ECC-pbuf and M-ECC-pfbuf have the same IPC performance but different energy consumption, only M-ECC-pbuf is shown here), almost all the benchmarks show severe performance degradation when using the M-ECC-only scheme, especially for those benchmarks with relatively higher L2 cache hit rate. The IPC performance degradation can be greatly compensated by using the M-ECC-pbuf (and M-ECC-pfbuf) scheme in which the predecoding buffer can very effectively reduce the occurrence of explicit multibit ECC decoding.

According to Table 4, the energy consumption of one M-ECC cache access is almost 20 times higher than that of one L2 cache core access. This clearly suggests that we must minimize the frequency of accessing the M-ECC cache, which is the purpose of introducing the FLU buffer. Fig. 10 shows the normalized power consumption of the entire M-ECC core using the M-ECC-pfbuf scheme with respect to the power consumption of the 1 MB L2 cache core. For the majority of benchmarks, the power consumption of the entire M-ECC core is less than 30 percent of that of the L2 cache core. As shown in Fig. 11, the energy consumption of L2 cache only occupies a quite small fraction with respect to the whole cache hierarchy (less than 11 percent). Hence, the energy consumption of the proposed M-ECC cache core is relatively insignificant.

Finally, according to the Cacti modeling results listed in Table 4, the silicon area of the M-ECC cache, predecoding buffer, and FLU buffer are much smaller than that of the L2 cache core, especially for the FLU buffer as it only contains the addresses of cache blocks. The total area overhead is less than 2.5 percent of the area of the 1 M 8-way associative L2 cache core. This clearly suggests that the proposed design is area-efficient.

5.2 Dirty Replication Cache

Fig. 12 shows the hit rates for the DR cache with 8, 16, 32, and 64 blocks, respectively, where the L2 cache is fixed as 1 MB

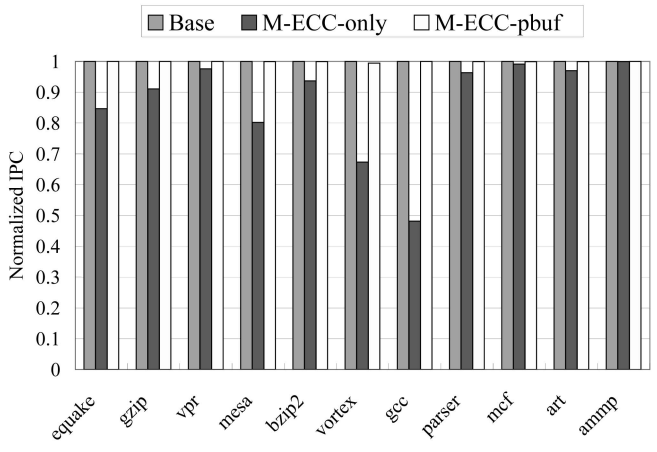


Fig. 9. Normalized IPC comparison among Base, M-ECC-only, and M-ECC-pbuf schemes.

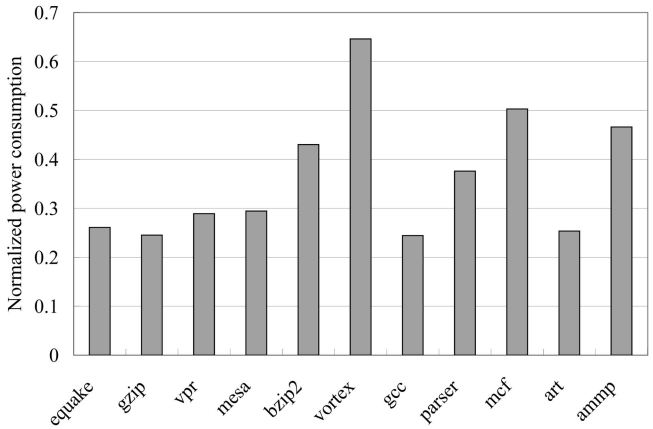


Fig. 10. Normalized power consumption of the M-ECC-pfbuf scheme.

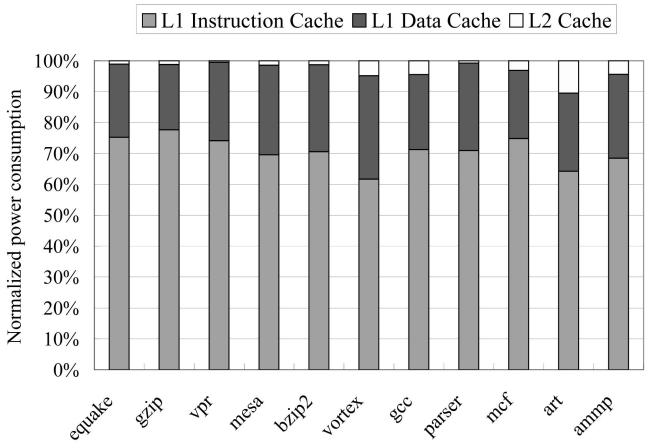


Fig. 11. Normalized power consumption comparison among L1 instruction cache, L1 data cache, and L2 cache.

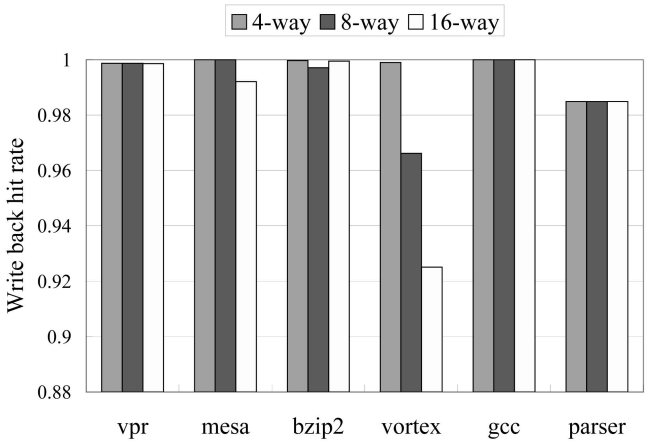


Fig. 13. Write-back hit rates for 4-way, 8-way, and 16-way set associative 1 MB L2 cache. The DR cache is fully associative with 64 blocks (only the six worst case benchmarks are shown).

8-way associative. For the DR cache with 64 blocks, the hit rate is above 96.4 percent for all the benchmarks. From the result of associative sensitivity experiment, shown in Fig. 13, the DR cache behaves quite consistently, except the benchmark "vortex" that has relatively very small amount of write back from L1 cache (i.e., 925 per million instructions).

Facilitated with the small dirty replication cache, a very high percentage of write-back data from L1 cache can be stored and replicated. In case the dirty replication cache is full, the least recent entry will be replaced and written back to the next level memory. Hence, when a soft error is detected, the processor can always recover the data from a backup in either the dirty replication cache or the next level memory. In the meantime, the proposed design tends to be more cost-efficient compared with other methods, including the AU-blk and AU-sblk schemes. Since AU-blk updates dirty data at cache block level, this scheme becomes an almost write-through policy at a high defect density, leading to a serious performance degradation. By moving to the cache subblock level, the AU-sblk scheme proposed in [5] can reduce the performance degradation; however, the area and power overhead tend to be high since each tag block in AU-sblk scheme adds extra 16 bits to realize the subblock-level assurance update. Our simulation based on Cacti shows that the energy per access to the modified L2 cache in the AU-sblk scheme is 0.0253 nJ and the area is $6.63\ \mathrm{mm}^2$. There are 26.5 percent and 11.0 percent increases, respectively. The area overhead of the DR cache is the same as the predecoding buffer, i.e., only 0.3 percent of the L2 cache area. As the energy consumption of the DR cache only occurs in the presence of L1 cache write back, the power is relatively much lower than the AU-sblk scheme. Since it is difficult to use Simplescalar to simulate how the L2-cache-to-main-memory communication impacts the overall performance, we do not show the IPC comparison. Nevertheless, it is clear that the DR cache tends to minimize the L2-cache-to-main-memory burden and may not lead to noticeable IPC degradation.

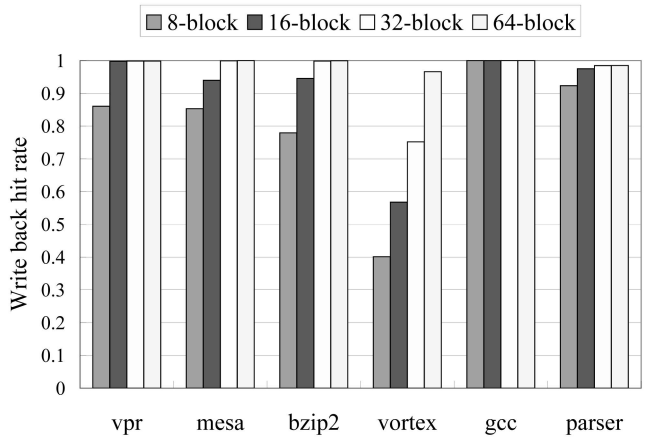


Fig. 12. Write-back hit rates for DR cache of 8, 16, 32, and 64 blocks. The L2 cache is fixed to be 1 MB 8-way associative (only the six worst case benchmarks are shown).

6 Conclusions

This work was motivated by the almost evident trend that on-chip SRAM will contain a large amount of parametric random cell defects if we want to leverage the future technology scaling to its full extent. In particular, we are interested in how to effectively use unconventional strong multibit ECC for SRAM-based L2 cache defect tolerance at minimal performance and implementation cost. This paper proposes an L2 cache architecture in which cache blocks containing more than one defect SRAM cells are protected by multibit ECC. Two small buffers, i.e., predecoding buffer and fast lookup buffer, are embedded into the L2 cache to largely reduce the probability that the major heavy-load multibit ECC functional blocks are explicitly invoked, hence reduce the performance degradation and energy cost incurred by the use of multibit ECC in L2 cache. Meanwhile, we propose to further embed a small dirty replication cache in L2 cache to ensure soft error tolerance by keeping copies of most recent write-back data from L1 data cache.

The proposed cache architecture has been extensively evaluated by using popular processor simulator and memory modeling tools. Results show that this proposed design solution is able to maintain almost the same IPC performance at the high defect density of 0.5 percent compared with the ideal defect-free L2 cache, while only incurring less than 2.5 percent of silicon area overhead and 36 percent of energy consumption overhead. With an area

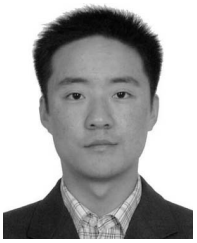
overhead of only 0.3 percent, the dirty replication cache can store up to 96.4 percent of the write-back data from L1 cache, which means it can ensure soft error tolerance for dirty cache blocks in L2 cache at very small overhead on L2-cache-to-main-memory communication load.

ACKNOWLEDGMENTS

This research was funded in part by grants from the Key Program of the National Natural Science Foundation of China (No. 60635050) and the National High Technology Research and Development Program of China (863 Program) (No. 2006AA01Z192). This work was done during the visit of H. Sun at Rensselear Polytechnic Institute.

REFERENCES

[1] Semiconductor Industry Assoc., *The Int'l Technology Roadmap for Semiconductors (ITRS),* http://www.itrs.net/reports.html, 2009.
[2] N. Quach, "High Availability and Reliability in the Itanium Processor," *IEEE Micro.,* vol. 20, no. 5, pp. 61-69, Sept. 2000.
[3] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach,* fourth ed. Morgan Kaufmann, 2006.
[4] A. Bhavnagarwala, S. Kosonocky, C. Radens, K. Stawiasz, R. Mann, Q. Ye, and K. Chin, "Fluctuation Limits & Scaling Opportunities for CMOS SRAM Cells," *Proc. IEEE Int'l Electron Devices Meeting,* pp. 659-662, 2005.
[5] L.D. Hung, H. Irie, M. Goshima, and S. Sakai, "Utilization of SECDED for Soft Error and Variation Induced Defect Tolerance in Caches," *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE),* pp. 1-6, 2007.
[6] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J.C. Hoe, "Multi-Bit Error Tolerant Caches Using Two-Dimensional Error Coding," *Proc. 40th Ann. ACM/IEEE Int'l Symp. Microarchitecture (Micro-40),* pp. 197-209, 2007.
[7] S. Lin and D.J. Costello, *Error Control Coding: Fundamentals and Applications,* second ed. Prentice Hall, 2004.
[8] C.L. Chen and M.Y. Hsiao, "Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review," *IBM J. Research and Development,* vol. 28, no. 2, pp. 124-134, Mar. 1984.
[9] K. Chakraborty and P. Mazumder, *Fault-Tolerance and Reliability Techniques for High-Density Random-Access Memories.* Prentice Hall, 2002.
[10] K. Pagiamtzis, "Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey," *IEEE J. Solid State Circuits,* vol. 41, no. 3, pp. 712-727, Mar. 2006.
[11] M. Nicolaidis, N. Achouri, and L. Anghel, "A Diversified Memory Built-In Self-Repair Approach for Nanotechnologies," *Proc. IEEE Very Large Scale Integration (VLSI) Test Symp.,* pp. 313-318, 2004.
[12] S. Wang and L. Wang, "Design of Error-Tolerant Cache Memory for Multithreaded Computing," *Proc. IEEE Int'l Symp. Circuits and Systems,* pp. 1890-1893, May 2008.
[13] G. Di Natale, A. Benso, S. Chiusano, and P. Prinetto, "An Online BIST RAM Architecture with Self-Repair Capabilities," *IEEE Trans. Reliability,* vol. 51, no. 1, pp. 123-128, Mar. 2002.
[14] *CACTI: An Integrated Cache and Memory Access Time, Cycle Time, Area, Leakage, and Dynamic Power Model,* http://www.hpl.hp.com/research/cacti/, 2009.
[15] S. Lin and D.J. Costello, *Error Control Coding: Fundamentals and Applications.* Prentice Hall, 1983.
[16] M.H. Tehranipour, Z. Navabi, and S.M. Falkhrai, "An Efficient BIST Method for Testing of Embedded SRAMs," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS),* pp. 73-76, May 2001.
[17] S. Nakahara, K. Higeta, M. Kohno, T. Kawamura, and K. Kakitani, "Built-In Self-Test for GHz Embedded SRAMs Using Flexible Pattern Generator and New Repair Algorithm," *Proc. Int'l Test Conf.,* pp. 301-310, 1999.
[18] L.D. Hung et al., "Mitigating Soft Errors in Highly Associative Cache with CAM-Based Tag," *Proc. IEEE Int'l Conf. Computer Design,* pp. 342-347, Oct. 2005.
[19] K. Pagiamtzis, N. Azizi, and F.N. Najm, "A Soft-Error Tolerant Content-Addresable Memory (CAM) Using an Error-Correcting-Match Scheme," *Proc. IEEE Custom Integrated Circuits Conf.,* pp. 301-304, 2006.
[20] H.-J. Lee, "Immediate Soft Error Detection Using Pass Gate Logic for Content Addressable Memory," *Electronics Letters,* vol. 44, no. 4, pp. 269-270, Feb. 2008.
[21] http://www.simplescalar.com, 2008.
[22] R.E. Blahut, *Algebraic Codes for Data Transmission.* Cambridge Univ. Press, 2003.
[23] Standard Performance Evaluation Corporation, http://www.spec.org, 2000.



Hongbin Sun received the BS degree in electrical engineering from Xi'an Jiaotong University, China, in 2003. He is currently a PhD student in the School of Electronic and Information Engineering at Xi'an Jiaotong University. He was a visiting scholar in the Electrical, Computer, and Systems Engineering Department at Rensselaer Polytechnic Institute, Troy, New York, from November 2007 to October 2008. His current research interests include fault-tolerant computer architecture, 3D memory-processor integration, and VLSI architecture for digital video processing.

Nanning Zheng graduated from the Department of Electrical Engineering, Xi'an Jiaotong University, China, in 1975, received the MS degree in information and control engineering from Xi'an Jiaotong University, in 1981, and received the PhD degree in electrical engineering from Keio University, Yokohama, Japan, in 1985. He joined Xi'an Jiaotong University in 1975, where he is currently a professor and the director of the Institute of Artificial Intelligence and Robotics. His research interests include computer vision, pattern recognition, machine vision and image processing, neural networks, and hardware implementation of intelligent systems. He became a member of the Chinese Academy of Engineering in 1999, and has been the chief scientist and the director of the Information Technology Committee of the China National High Technology Research and Development Program since 2001. He was general chair of the International Symposium on Information Theory and Its Applications and general co-chair of the International Symposium on Nonlinear Theory and Its Applications, both in 2002. He is a member of the Board of Governors of the IEEE ITS Society and the Chinese representative on the Governing Board of the International Association for Pattern Recognition. He also serves as an executive deputy editor of the *Chinese Science Bulletin*. He is a fellow of the IEEE.

Tong Zhang received the BS and MS degrees in electrical engineering from Xi'an Jiaotong University, China, in 1995 and 1998, respectively. He received the PhD degree in electrical engineering from the University of Minnesota, Minneapolis, in 2002. Currently, he is an associate professor in the Electrical, Computer, and Systems Engineering Department at Rensselaer Polytechnic Institute, Troy, New York. His current research interests include algorithm and architecture co-design for communication and data storage systems, variation-tolerant signal processing IC design, fault-tolerant system design for digital memory, and interconnect system design for hybrid CMOS/nanodevice electronic systems. Currently, he serves as an associate editor for the *IEEE Transactions on Circuits and Systems—II* and the *IEEE Transactions on Signal Processing*. He is a senior member of the IEEE.