Regular Paper

Zigzag-HVP: A Cost-effective Technique to Mitigate Soft Errors in Caches with Word-based Access

Luong Dinh Hung,† Masahiro Goshima† and Shuichi Sakai†

Error Correction Code (ECC) is widely used to detect and correct soft errors in VLSI caches. Maintaining ECC on a per-word basis, which is preferable in caches with word-based access, is expensive. This paper proposes Zigzag-HVP, a cost-effective technique to detect and correct soft errors in such caches. Zigzag-HVP utilizes horizontal-vertical parity (HVP). HVP maintains the parity of a data array both horizontally and vertically. Basic HVP can detect and correct a single bit error (SBE), but not a multi-bit error (MBE). By dividing the data array into multiple HVP domains and interleaving bits of different domains, a spatial MBE can be converted to multiple SBEs, each of which can be detected and corrected by the corresponding parity domain. Vertical parity updates and error recovery in Zigzag-HVP can be efficiently executed through modifications to the cache data paths, write-buffer, and Built-In Self Test. The evaluation results indicate that the area and power overheads of Zigzag-HVP caches are lower than those of ECC-based ones.

1. Introduction

Radiation particle strikes cause soft error problems. Therefore, measures must be employed in processors to ensure greater reliability against them. Since most of the processor resources are allocated for caches, making the caches resilient to soft errors is important.

Error Correction Code (ECC) — specifically Single Error Correcting and Double Error Detecting Hamming Code (SECDED) — is widely used to detect and correct soft errors in caches. ECC maintains the check bits per unit of data. **Table 1** lists the number of check bits and the SECDED overheads for various unit sizes.

Maintaining ECC for a large unit is preferable to keep the overhead low. However, for caches with word-based access (e.g., L1 data caches or L2 caches with write-through L1 caches), maintaining ECC on a per-word basis is preferred [6),12)]. Otherwise, partial updates to the larger ECC units will frequently occur, incurring expensive *read-modify-write* operations that read the entire units, recompute the check bits, and write back the units. However, the cost of maintaining ECC per-word is high. For instance, a 32-bit word requires seven SECDED check bits and incurs a 22% area overhead.

This paper proposes Zigzag-HVP — a low-cost technique to detect and correct soft errors for these word-based accessed caches. The technique makes use of horizontal-vertical parity (HVP) [10)]. HVP maintains the parity of the data array both horizontally and vertically. Basic HVP can detect and correct a single bit error (SBE), but not a multi-bit error (MBE). The probability that multiple errors will be accumulated from multiple particle strikes is very low in frequently accessed caches. Instead, MBE is mostly caused by a single particle strike that corrupts multiple bits at once [11)]. Therefore, the corrupted bits are located close to one another. By dividing the data array into multiple HVP domains and interleaving bits of different domains, a spatial MBE is converted into multiple SBEs, each of which can be detected and corrected by the corresponding domain.

An error recovery routine is executed when an error is detected by horizontal parity. By sequentially reading the data words belonging to the parity domain, vertical parity can be recomputed to locate the error bit. The Built-In Self Test (BIST) hardware [1),3)] is enhanced to include such a recovery function. We also modify the cache data path and write buffer to efficiently accommodate the vertical parity updates. The evaluation results indicate that the area and power overheads of Zigzag-HVP caches are lower than those of ECC-based ones.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 explains the basic concept of HVP and its limitations. Section 4 describes Zigzag-HVP. Section 5 discusses the applications of Zigzag-HVP. Section 6.1 presents the evaluation re-

† Graduate School of Information Science and Technology, The University of Tokyo

Table 1 Number of SECDED check bits for various data unit sizes.

Data unit size (bits)	16	32	64	128	256
No. of check bits	6	7	8	9	10
Overhead (%)	37.5	21.9	12.5	7.0	3.9

sults. Finally, Section 7 concludes the paper.

2. Related Work

The use of a small cache to store check bits or replicas of recently used data has been proposed [9]. Replication cache [20] enhances this concept by using a small fully associative cache to store the replica of every write to the L1 cache. ICR cache [21] uses those cache blocks that are predicted to be "dead" as replicas of "hot" dirty data. While these techniques offer lower overheads than ECC, a large portion of the dirty data still remains unprotected if the locality of the data is low. The portion of unprotected data for some benchmarks has been as high as 40% [9], 30% [21], or 5% [20]. The level of dependability achieved by these techniques is obviously not acceptable for applications demanding extremely high reliability. In addition, while a replication cache [20] may be suitable for small L1 caches, its application to large L2 caches is impractical. Zigzag-HVP can reduce the error rate by many orders of magnitude and can be applied to both L1 and L2 caches.

Cross-parity [14] can deal with MBEs by maintaining diagonal parity, in addition to horizontal and vertical. However, an examination of the horizontal parity alone cannot expose the existence of some MBEs (e.g., MBEs where the number of error bits in the same row is even). Recomputation and checking of vertical or diagonal parity are required to detect such MBEs. However, these operations are expensive and it is impractical to execute them on every data access. By relying on interleaving to disperse the error bits, the existence of spatial MBEs in Zigzag-HVP can be detected by only examining the horizontal parity and the vertical parity only needs to be recomputed after the errors have been detected.

The concept of interleaving has been used to combat burst errors. Existing interleaving schemes usually require intensive computation to detect and correct the errors. However, on-chip caches are latency-critical, making these complex interleaving schemes unacceptable. Simple interleaving of ECC units in the same row to tolerate MBEs has been used in caches or memories [13),18]. Zigzag-HVP interleaves the data in two dimensions and its superior practicality is explained in this paper.

$$\begin{array}{cccc|c} d_{1|1} & d_{1|2} & \dots & d_{1|n} & hp_1 \\ d_{2|1} & d_{2|2} & \dots & d_{2|n} & hp_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ d_{m|1} & d_{m|2} & \dots & d_{m|n} & hp_m \\ \hline vp_1 & vp_2 & \dots & vp_n & hvp \end{array}$$

$$\begin{aligned} hp_i &= d_{i|1} \oplus d_{i|2} \oplus \ldots \oplus d_{i|n} \\ vp_j &= d_{1|j} \oplus d_{2|j} \oplus \ldots \oplus d_{m|j} \\ hvp &= vp_1 \oplus vp_2 \oplus \ldots \oplus vp_n \\ &= hp_1 \oplus hp_2 \oplus \ldots \oplus hp_m \end{aligned}$$

$\oplus$ denotes Exclusive OR

Fig. 1 Horizontal-vertical parity.

3. Horizontal-Vertical Parity and Multi-Bit Errors

3.1 HVP Concept

Figure 1 shows the concept of HVP [10]. The parity of the $m \times n$ data array is maintained both horizontally and vertically. $d_{i|j}$ denotes a data bit in the i-th row and j-th column of the array. hp_i and vp_j are the parity bits of the i-th row and j-th column, respectively. hvp is the sum parity of the vertical parity bits, and this is also equal to the sum parity of the horizontal parity bits.

HVP can detect and correct SBEs. Assume that the bit $d_{i|j}$ is corrupted. When row i is accessed, the parity of the row is recomputed and compared to hp_i. A mismatch indicates the existence of an error in the row. The vertical parity bits are then recomputed by sequentially reading all rows. The resulting vertical parity bits are compared to those currently stored in the array. A mismatch in column j of the vertical parity indicates the position of the error in the victim row.

For an $m \times n$ data array, HVP requires $m+n$ check bits or an $(m+n)/(m \times n)$ area overhead. The overhead is small when m and n are sufficiently large.

3.2 Limitations with Basic HVP

While the basic HVP scheme previously described can detect and correct an SBE, it may be unable to detect and correct an MBE.

Figure 2 shows several cases of MBEs in a 4×4 data array. In case A, since the corruption of both $d_{2|1}$ and $d_{2|2}$ leaves the parity of row 2 unchanged, horizontal parity is unable to detect the existence of the errors. In case B, while the

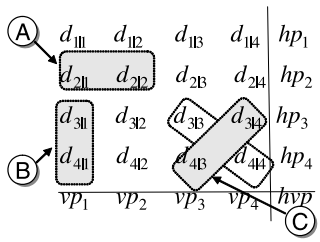

Fig. 2 Examples of multi-bit errors.

existence of errors in row 3 and row 4 can be detected by hp_3 and hp_4, the exact positions of the errors in the rows cannot be revealed by vertical parity since the vertical parity of column 1 remains unchanged. In case C, while horizontal parity reveals the existence of errors in rows 3 and 4 and vertical parity reveals the existence of errors in columns 3 and 4, HVP still cannot determine which one of the two pairs $(d_{3|4}, d_{4|3})$ or $(d_{3|3}, d_{4|4})$ is corrupted.

3.3 Multi-Bit Errors and Characteristics

An MBE results from the error bits accumulated from multiple particle strikes, or the error bits caused by a single particle strike. We call the former a *multi-strike multi-bit error* (MS-MBE), and the latter a *single-strike multi-bit error* (SS-MBE).

A particle strike occurs and corrupts a single bit in an MS-MBE. Another strike occurs and corrupts a different bit before the erroneous data are accessed, through which the error could have been detected and corrected. An MS-MBE could pose a problem in very large memories where the data might not be accessed for a long time and the probability that undetected errors will accumulate cannot be ignored.

A single strike corrupts multiple bits at once in an SS-MBE. In contrast to MS-MBE, the error bits in SS-MBE are *closely* located. The scaling trend towards smaller device dimensions and lower supply voltages has increased the probability that a strike will result in an SS-MBE [7),11)].

4. Zigzag-HVP

Since on-chip caches usually have a high access frequency, an error generated by a strike is likely be detected and corrected before the next strike occurs. The results in Section 6.1 confirm that the MS-MBE rate in caches is very low. We therefore focus on the measures against SS-MBE.

Zigzag-HVP exploits the property that error bits in SS-MBE are closely located. Zigzag-HVP groups the data bits into multiple *parity domains*. Each parity domain consists of several data words and is protected by HVP. By interleaving different parity domains, the spatial error bits in an SS-MBE are converted to multiple SBEs. Each SBE belongs to a parity domain that can be successfully detected and corrected.

Let us define some terminology. A cache can be expressed as a data array with N_R rows. Each row contains N_L cache lines, each line contains N_W words, and each word contains N_B bits. The cache size is $N_R \times N_L \times N_W \times N_B$ bits.

4.1 Bit Interleaving Scheme

SS-MBE can have multiple error bits in the same row, or the same column [11)]. We will now describe the interleaving scheme to deal with these kinds of errors.

4.1.1 Dealing with Horizontal MBE

Interleaving layout of words converts adjacent error bits in the same row into SBEs in different words. ECC-protected caches have used this technique to effectively reduce MBEs on the same ECC unit [13),18)]. We use the same technique in Zigzag-HVP to deal with horizontal MBE. The interleaved words in a row belong to different parity domains. **Figure 3**-a illustrates a simple example; two words are interleaved in each row and up to two bit errors can be tolerated (b_i^j indicates the j-th bit of the i-th word in a row).

We must interleave at least d different words to tolerate up to d bit errors. In a typical SRAM design, several neighbor bits in the same row share a sense amplifier, from which only one bit is multiplexed and amplified in an access. We have two possible options in choosing how to interleave words horizontally. If the number of cache lines in each row N_L is larger than or equal to d, we should interleave words from different cache lines. In this case, the words belonging to a cache line can be accessed at once, which is preferred for line-based cache operations such as line replacements or cache refills. If N_L is smaller than d (possible in small caches), the interleaved word could be selected from the same line. In this case, reading a whole cache line requires several cache accesses since an access can only read a subset of words of the cache line.

The number of horizontal parity bits is equal to the number of cache words $(N_R \times N_L \times N_W)$

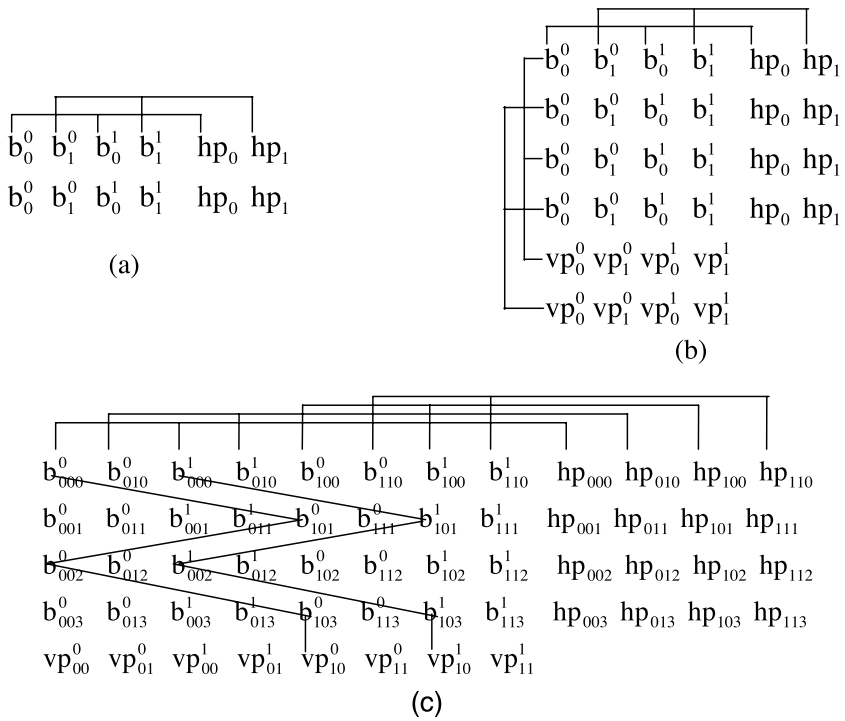

Fig. 3 Interleaving schemes to deal with MBEs. Bits are interleaved to tolerate horizontal MBEs in (a). Scheme in (b) tolerates horizontal & vertical MBEs but requires many vertical parity bits. Scheme in (c) can tolerate horizontal & vertical MBEs with fewer vertical parity bits.

and is independent of how the words are interleaved.

4.1.2 Dealing with Vertical MBE

Vertical MBE can also be dealt with by using the same interleaving concept as in the case of horizontal MBE. Consecutive bits in the same column are protected by different vertical parity bits. In Fig. 3-b, bits in the even and odd rows are protected by a different vertical parity and up to two bit errors can be tolerated in this example. This scheme requires d vertical parity bits in each column to tolerate up to d bit errors. Since the vertical parity bits are frequently updated, they should be implemented as flipflops or latches but this would use larger areas than normal SRAM cells. The overheads for vertical parity increase with a large d and may offset the benefit of HVP.

We instead propose an original scheme for encoding vertical parity. The HVP domain is constructed so that a vertical bit is calculated from bits located in a *zig-zag* path, rather than from bits in the same column. The physical locations of any two bits in the same zig-zag path are separated by a sufficient distance so that both cannot be corrupted by a particle strike. The number of vertical parity bits in this scheme is equal to the number of columns and is *independent* of d.

To formalize the scheme, the expression w_{ijk} ($0 \leq i < N_W$, $0 \leq j < N_L$, $0 \leq k < N_R$) is used to refer to the i-th word of the j-th cache line of the k-th row. A parity domain $\mathcal{PD}_{mn}$ ($0 \leq m < N_W$, $0 \leq n < N_L$) is a set of words that can be expressed by Eq. (1).

$$\mathcal{PD}_{mn} = \{w_{inj} |\ (i - j) \equiv m\ (mod\ N_W)\} \tag{1}$$

Figure 3-c illustrates a simple example. The data array has four rows, each row contains two cache lines, and each cache line has two two-bit words. b^l_{ijk} indicates the l-th bit of word w_{ijk}. The two cache lines in each row are interleaved to tolerate horizontal MBEs. Two zigzag paths which consist of bits belonging to the same parity domain ($\mathcal{PD}_{00}$) are shown in the figure. The parity domain consists of four words: w_{000}, w_{101}, w_{002}, and w_{103}. MBEs having up to two bits can be tolerated in this example.

The proposed scheme can tolerate up to an N_W-bit vertical MBE. Given that a cache line typically consists of 8∼32 words and an SS-MBE contains no more than four bits in current processes [11], our zigzag scheme can effectively deal with vertical MBEs.

4.1.3 Other MBE

If the interleaving scheme in Section 4.1.1 tolerates up to d_1 consecutive error bits in a row and the interleaving scheme in Section 4.1.2 tolerates up to d_2 consecutive error bits in a column, then any MBE in which the error bits are confined to a $d_2 \times d_1$ array can be successfully detected and corrected.

4.2 Parity Update Mechanism

Any update of a word (e.g., in processor writes or cache line replacements) requires the parity of the corresponding domain to be updated. From Eq. (1), the parity domain that a word belongs to can easily be determined from the values of the bits used to index the row, and the values of the bits used to index the word inside the cache line.

Updating the horizontal parity is simple; i.e., the parity bit is newly calculated from the updated word value and stored together with the word. Updating the vertical parity follows Eq. (2).

$$\mathcal{VP}_{new} = \mathcal{VP}_{old} \oplus w_{old} \oplus w_{new} \qquad (2)$$

The new vertical parity of the domain ($\mathcal{VP}_{new}$) is the exclusive-OR of the old vertical parity ($\mathcal{VP}_{old}$), old and new values of the data word (w_{old} and w_{new}). The horizontal parity bits are included in the w_{old} and w_{new}.

The update of the vertical parity requires the old value of the word. Section 5 discusses how caches can be modified to effectively supply the old words.

Updating vertical parity can be done in parallel with writing the data word into data array. Therefore, the access latency of a Zigzag-HVP cache is comparable to that of a cache protected by a simple parity.

4.3 Error Recovery

When a word is read, the horizontal parity bit is recomputed and compared with the prestored value. A mismatch indicates the existence of a bit error in the word. A dedicated error recovery routine is then triggered. Since there is a possibility that bit error(s) may also be present in other word(s), the routine examines not only the parity domain containing the identified erroneous word but also all the parity domains. For each domain, the routine sequentially reads all the words belonging to that domain. The horizontal parity of each word and the vertical parity of the domain are recomputed, and compared with the old ones. If an SBE is present in the domain, its location can be determined.

Modern caches are typically equipped with a Built-In Self Test (BIST) [1]. The BIST accesses data in particular access patterns, also called *marching patterns*, to locate potential manufacturing defects. Modern BISTs are programmable and support various marching patterns [3]. The capability of sequentially reading the words belonging to a parity domain can be supported by extending the existing BIST hardware. The address patterns of those data words belonging to the same parity domain are derived from Eq. (1). By accommodating such patterns into the BIST, an error recovery routine can be achieved at modest hardware cost.

Recovering from errors requires the cache to be fully scanned. Nevertheless, the overhead for error recovery is small since soft errors occurs very infrequently. For instance, let us consider a 512 KB cache. Assuming a per-word access throughput of 512 M-word/sec, a full scan of all parity domains in the cache requires 269 μsec. Such an overhead is incurred once every 17 years and is therefore negligible (refer to Section 6.5 for the cache error rate).

5. Applications of Zigzag-HVP

Two possible candidates for Zigzag-HVP are 1) L1 write-back caches and 2) L2 caches with write-through L1 caches. This section focuses on how the data paths of these caches can be modified so that the vertical parity update of Zigzag-HVP can be efficiently executed.

5.1 L1 Write-back Caches

Before writing a word, the L1 cache needs to probe its tag to determine whether it is holding the word or not. The L1 cache is adapted so that data access occurs in parallel with the tag probing. If the line is found in the L1 cache, the old value of the data word is accessible after the tag probe phase. The cache then proceeds to write the new value while the old value is passed to the vertical parity update unit. Such a modification is easily accomplished since, in practice, L1 caches already perform tag probe and data access in parallel to achieve minimum latency.

Write-back caches usually employ a *write-*

allocate policy [8]: A write miss fetches the missed line from the lower level cache and allocates the location for storing the line. In the case of a write miss, the vertical parity update unit will receive the old value of the modified word when the line is retrieved from the lower level cache.

5.2 L2 Caches with Write-through L1 Caches

Many processors adopt a cache hierarchy in which the L1 data cache is write-through and is backed by a large L2 write-back cache. Making the L1 data cache a write-through cache simplifies the task of maintaining cache coherency in multiprocessor systems [5]. Maintaining simple parity for the detection of errors is sufficient for write-through L1 caches since correct data can be obtained from the L2 caches. The application of Zigzag-HVP to the L2 caches is discussed after this.

5.2.1 Write Buffer and Its Implications:

A processor with a write-through L1 cache usually includes a write buffer. The write buffer can coalesce writes to the same cache block, thereby reducing the traffic to the L2 cache. One could maintain ECC in large data units (e.g., double-word (64 bits) [15], or per quad-word (128 bits) [19]) to reduce the hardware overheads, and rely on the write buffer to merge updates to consecutive words into a single update of a large ECC word so that the read-modify-write operations could be reduced. However, our experiments confirmed that even with the write buffer having sufficient entries, a majority of ECC words are still partially written back.

Per-word ECC incurs a large hardware cost, while a larger ECC unit incurs frequent read-modify-write operations even with the presence of the write buffer. Zigzag-HVP can deal with such shortcomings with ECC schemes.

5.2.2 Support for Efficient Vertical Parity Updates:

The old values of the modified words, which are required for vertical parity updates in Zigzag-HVP, can be supplied directly by the L2 cache. However, since the L2 cache is large, reading the old values from L2 caches incurs a large power overhead.

The old words can be supplied by the L1 cache instead of being supplied from the L2 cache. Before writing a word, the L1 cache must probe its tag to determine whether it is holding the word or not. The L1 cache is adapted so that data access occurs in parallel with tag probing. If the write hits in the L1 cache, the old value of the data word is accessible and passed to the vertical parity update unit of the L2 cache. Reading the data word from a small L1 cache consumes less power than reading from a large L2 cache. The modified data path is shown in **Fig. 4**-a.

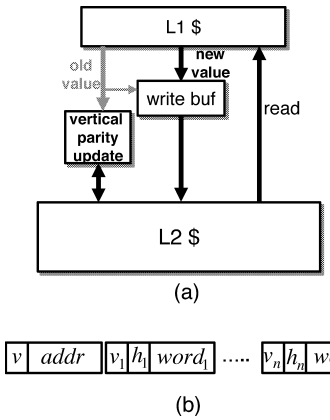

Fig. 4 Modified data path of L2 cache with write-through L1 cache and write buffer in (a). An entry of write buffer in (b).

Let us consider the case in which a write miss occurs in the L1 cache. Write-through caches usually employ a *no-write-allocate* policy: The missed line is not allocated in the L1 cache. When a write miss occurs in the no-write-allocate L1 cache, the old value of the updated word is explicitly supplied by the L2 cache to the vertical parity update unit. Since the majority of writes hit in the L1 cache, the frequency of access to the L2 cache to obtain the old values is low.

The write buffer is also modified. Figure 4-b shows an entry of the write buffer. The v bit, if set, signifies that the entry is valid and it is holding the words addressed by the *addr* field. The v_i bit indicates whether the i-th word is valid or not. A new bit h bit is added to each word. The h_i bit, if set, indicates that the i-th word did hit in the L1 cache and that the vertical parity update unit has already been updated with the old value of the word. When the entry is retired, the valid words are written back to the L2 cache. For those valid words for which h bits are not set, the old values must be explicitly read from L2 and passed to the vertical parity update unit. When an error is detected in the L2 cache, the write buffer must write

back all valid entries to the L2 cache before the error recovery can take place.

6. Evaluation

This section evaluates the application of Zigzag-HVP to L2 caches with write-through L1 caches. Compared to L1 caches, L2 caches are typically much larger and the costs of implementing protection against soft errors are also higher; therefore, the reduction of such costs is preferred. The L2 caches also present other interesting considerations due to the presence of write buffers.

6.1 Evaluation Methodology

The L2 cache used in the evaluation is a unified, 512-KB, 64 B-line, four-way set associative cache. It is accompanied by a 16 KB, write-through L1 data cache, and an eight-entry write buffer. The write buffer attempts to retire the oldest entry whenever more than six entries have been occupied (*retire-at-6* policy [17]). The detail configuration of the evaluated architecture is shown in **Table 2**.

While 64-bit processors are increasingly gaining popularity, there are many 32-bit processors and software built on 32-bit architectures still in use in practice. Even in 64-bit processors, support for efficient execution of 32-bit software is essential. For instance, the L2 caches in Itanium processors maintain ECC per 32-bit data to accommodate frequent 32-bit data updates encountered when executing 32-bit software [6]. A 32-bit word size is assumed in the evaluation.

Four caches with different error protection schemes are assumed in the evaluation:

- **NOPRT:** L2 cache without any soft error protection.
- **ECCSW:** L2 cache in which both the tag and data portions are protected by SECDED per single-word.
- **ECCQW:** L2 cache in which both the tag and data portions are protected SECDED per quad-word.
- **ZHVP:** L2 cache in which the tag and data portions are protected by Zigzag-HVP.

We evaluate the unrecoverable error rate, number of check bits, and power consumption. Cache access activity, which is required for calculating the error rate and power consumption, is collected from cycle-accurate processor simulation using SimpleScalar toolset [4]. SPEC2000 benchmarks are used in the evaluation. Each benchmark is run for four billion instructions. Cacti tool [16] is used to determine the physical configurations of the L2 caches.

Table 2 Parameters of simulated architecture.

Processor Parameters	
Frequency	1 GHz
Functional Units	4 integer ALUs, 4 FP ALUs
	1 integer multiplier/divider
	1 FP multiplier/divider
LSQ size	8 instructions
RUU size	16 instructions
Issue Width	4 instructions/cycle
Memory Hierarchy Parameters	
L1 i-cache	16 KB, direct-map, 32 B block
	1 cycle latency
L1 d-cache	16 KB, 4-way, 32 B block
	1 cycle latency
	write-through & no-write-allocate
Write buffer	eight 32 B entries
	retired-at-6
L2	512 KB, unified, 4-way
	64 B block
	6 cycle latency
	write-back
Memory	100 cycle latency

6.2 Physical Configurations of L2 caches

Soft error experiments with SRAM fabricated in 130 nm and 90 nm processes confirmed the existence of up to four-bit MBEs [11]. The implementation of protection schemes to tolerate up to four bit SS-MBEs is considered in the evaluation.

The L2 cache contains 2,048 sets. Each set has four 64-B cache lines. Each tag entry has 19 bits (15 tag bits and four status bits). Cacti tool suggests that the tag portion should be divided into 256 rows; each row containing 32 tags from eight sets. Interleaving the tags of the different sets in the same row together tolerates horizontal MBE while allowing four tags of the same set to be read simultaneously in a cache access. Each tag is an ECC unit in ECCSW and ECCQW. For ZHVP, the tag portion consists of 32 parity domains and each domain is a 256×19 bit array.

Cacti similarly suggests the data portion to be an array of 2,048 rows; each row holds four cache lines of the same set. The tag portion and the data portion of a L2 cache are accessed sequentially to attain low power consumption. Thus, while four tags of the same set are required to be read simultaneously, only the data for the hitting line are read from the data portion. Therefore, four lines of the same set can be interleaved to tolerate MBEs without compromising the latency of line-based access. In ECCSW, each word in the data portion is an

Table 3 Number of check bits.

Protection Scheme	Number of check bits (Kb)	Overhead (%)
NOPRT	0	0
ECCSW	944	22.2
ECCQW	336	7.9
ZHVP	138.6	3.3

ECC unit. The ECC unit in ECCQW is comprised of four consecutive words of the same line. For ZHVP, the data portion consists of 64 parity domains; each domain is a 2,048 ×32 bit array.

6.3 Area Overhead

Table 3 lists the overhead in terms of the number of check bits required to implement the protection schemes. For ECCSW, each tag requires six check bits and each data word requires seven check bits, resulting in 944 Kb of check bits in total, or an 22.22% overhead. For ECCQW, each tag requires six check bits and nine check bits are required for every four words, resulting in 336 Kb of check bits in total, or an 7.91% overhead.

In ZHVP, the tag and data portions respectively require 8.6 and 130 Kb of check bits or an 3.26% overhead in total. The overhead of ZHVP is definitively smaller than those for ECCSW and ECCWQ.

We implemented a BIST similar to the one described in Ref. 3). It is synthesized using the Hitachi 0.18 μm process. The original BIST occupies 0.28% the area of the L2 cache. The BIST is then extended to support the error recovery function. The modified BIST occupies 0.35% the area of the cache. Therefore, the cost of implementing the error recovery is very small.

6.4 Power Overhead

The L1 data cache needs to supply the old values to the vertical parity update unit of the L2 cache when implementing Zigzag-HVP to the L2 cache. This increases the power consumption of the L1 cache. Power consumed by both the L1 data cache and the L2 cache are taken into account in the evaluation.

The power consumption of individual accesses to the L1 and L2 caches are computed with Cacti. We modify Cacti to allow the power consumption to be computed based on the granularity of the accessed data. We implemented 32-bit SECDED, 128-bit SECDED, and horizontal vertical parity calculation circuits in the 0.18 μm process and then used Synopsis NanoSim to calculate their power consumption.

Figure 5 breaks down the power consumed in the L1 data cache and L2 cache for the benchmarks. ECCSW increases the power consumption in the L2 cache by 28% on average, mainly due to the power consumed by accessing the check bits in the tag and data portions. ECCQW increases the power consumption in the L2 cache by 80%, of which 54% is consumed by reading the partially-modified quad-words and the remaining 26% is consumed by accessing and computing the check bits. The power consumption of the L1 cache remains unchanged for ECCSW and ECCQW.

Reading the old values of the updated data words for ZHVP increases the power consumption of the L1 cache by 18%. We confirmed that 94.4% of writes hit in the L1 cache. Additional access to the L2 cache to obtain the data words missed in the L1 cache and parity calculation increases the power consumption of the L2 cache by 4%.

When both the L1 cache and L2 cache are taken into account, ECCSW, ECCQW, and ZHVP respectively increase the total power by 17%, 49%, and 10%. ZHVP therefore consumes less power than ECC-based schemes.

6.5 Unrecoverable Soft Error Rate

We assume that the soft error rate of an unprotected SRAM (*SER*) equal to 1.6 KFIT[☆] per megabit [2)] and soft errors follow a uniform distribution. We will now describe the mechanism for calculating the unrecoverable soft error rates (*URSER*) of the L2 caches.

Any error in NOPRT results is unrecoverable; thus the *URSER* in this case is equal to $SER \times cachesize$.

Unrecoverable errors in caches other than NOPRT result from errors accumulated from multiple strikes. **Figure 6** shows two successive accesses to the same data unit. The second access is a read access, following the first after a delay of T. Two strikes occurring in the interval between the two accesses result in unrecoverable errors in the data unit, and such a possibility is equal to $P_1(T) \times P_2(T)$, where $P_1(T)$, $P_2(T)$ are the probabilities of the two strikes occurring in a time interval T. For ECCSW, the data unit in consideration is a data word, and $P_1(T)$, and $P_2(T)$ are the probabilities of strikes occurring in the same data word ($P_1(T) = P_2(T) = SER \times T \times wordsize$). Similarly, the

☆ One FIT (Failure In Time) corresponds to one failure per 10^9 hours

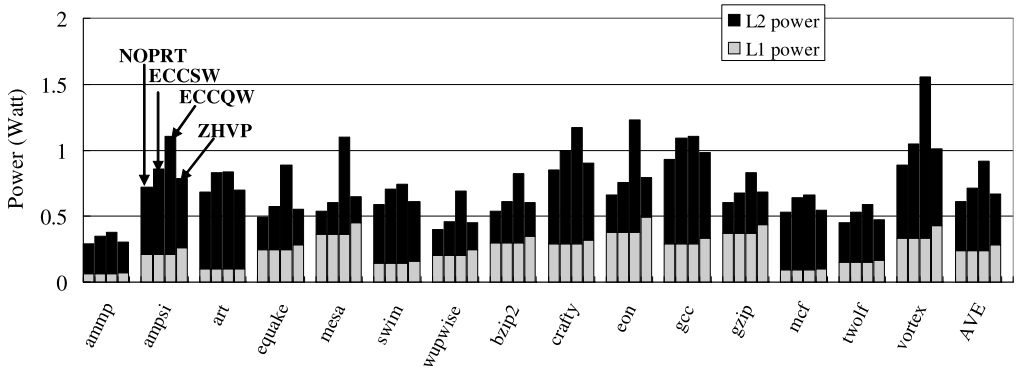

Fig. 5 Breakdown of power consumption.

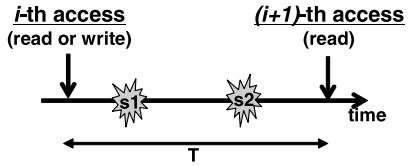

Fig. 6 Two strikes between two data accesses result in unrecoverable error.

Table 4 Unrecoverable soft error rate.

Protection Scheme	*URSER*(KFIT)
NOPRT	6.63
ECCSW	$1.72 * 10^{-16}$
ECCQW	$5.85 * 10^{-16}$
ZHVP	$2.37 * 10^{-13}$

data unit is a quad-word, and $P_1(T)$ and $P_2(T)$ are equal to $SER \times T \times quadwordsize$ for ECCQW.

In ZHVP, the data unit is a single word. Unrecoverable errors result if one strike occurs in the data word ($P_1(T) = SER \times T \times wordsize$), and the other strike occurs in the same parity domain with the data word in consideration ($P_2(T) = SER \times T \times paritydomainsize$ where $paritydomainsize$ is the number of bits in a parity domain).

Cache access activity for each data unit collected from the cycle-accurate processor simulation allows us to compute the *URSER* of each data unit. The $USRER$ of the entire cache is the sum of the *URSER* of all individual data units.

Table 4 lists the *URSER* of the L2 caches, averaged for all benchmarks. NOPRT's *URSER* is 6.63 KFIT, or roughly one error in 17 years. Such an *URSER* would be unacceptable since a parallel system consisting of 1024 processor nodes must fail every week.

The protection schemes other than NOPRT achieved great reduction in *URSER*. While ECCSW and ECCQW have lower *URSER*, the level of *URSER* achieved by ZHVP is clearly sufficient for all practical purposes. More specifically, ZHVP's *URSER* is equivalent to one failure in one million products about every 500 million years.

7. Conclusion

VLSI caches must employ measures against soft errors to ensure greater reliability. Maintaining ECC on a per-word basis, which is preferable for word-based accessed caches, is expensive. This paper presents Zigzag-HVP, an alternative technique to ECC to detect and correct the soft errors in such caches. Two-dimensional interleaving of data words converts a spatial MBE to multiple SBEs, each of which can be successfully detected and corrected. Modifications to the cache data paths, write buffer, and BIST allow parity updates and error recoveries to be executed efficiently. The implementation of Zigzag-HVP in a 512-KB L2 cache indicates that the respective overheads in terms of area and power consumption are 3.3% and 10%, which are smaller than those of the ECC-based ones. While Zigzag-HVP is vulnerable to MBEs accumulated from multiple particle strikes, our results demonstrate that such a probability would be extremely small.

Acknowledgments This research was partially supported by Grant-in-Aid for Fundamental Scientific Research B(2) #13480077, B(2) #16300013 from the Ministry of Education, Culture, Sports, Science and Technology Japan, a CREST project of the Japan Science and Technology Corporation, and by a 21st century COE project of the Japan Society for the Promotion of Science.

References

1) Adams, R.D.: *High Performance Memory Testing*, Kluwer Academic Publishers (2003).
2) Baumann, R.: Soft Errors in Advanced Computer System, *IEEE Design and Test of Computers*, Vol.22, No.3, pp.258–266 (2005).
3) Brauch, J. and Fleischman, J.: Design of Cache Test Hardware on the HP PA8500, *IEEE Design and Test of Computers*, Vol.15, No.3, pp.58–63 (1998).
4) Burger, D. and Austin, T.: The SimpleScalar Tool Set, Technical Report CS-TR-1997-1342, University of Wisconsin-Madison (1997).
5) Culler, D.E., Singh, J.P. and Gupta, A.: *Parallel Computer Architecture: A hardware/software approach*, Morgan Kaufman Publishers, Inc. (1999).
6) Grutkowski, T. and Riedlinger, R.: The High Bandwidth, 256 KB 2nd Level Cache on an Itanium Microprocessor, *Proc. IEEE International Symposium on Solid-State Circuits Conference*, pp.418–419 (2002).
7) Johansson, K., Ohlsson, M., Olsson, N., Blomgren, J. and Renberg, P.U.: Neutron Induced Single-word Multiple-bit Upset in SRAM, *IEEE Transactions on Nuclear Science*, Vol.46, No.6, pp.1427–1433 (1999).
8) Jouppi, N.P.: Cache Write Policies and Performance, *Proc. 20th International Symposium on Computer Architecture*, pp.191–201 (1993).
9) Kim, S. and Somani, A.K.: Area Efficient Architectures for Information Integrity in Cache Memories, *Proc. 26th International Symposium on Computer Architecture*, pp.246–255 (1999).
10) Lala, P.K.: *Self-checking and Fault-tolerant Digital Design*, Morgan Kaufman Publishers (2001).
11) Maiz, J., Hareland, S., Zhang, K. and Armstrong, P.: Characterization of Multi-bit Soft Error events in advanced SRAMs, *Proc. IEEE International Electron Devices Meeting*, pp.519–522 (2003).
12) Meaney, P.J., Swaney, S.B., Sanda, P.N. and Spainhower, L.: IBM z99 Soft Error Detection and Recovery, *IEEE Transactions on Device and Materials Reliability*, Vol.5, No.3, pp.419–427 (2005).
13) Osada, K., Saitoh, Y. and Ishibashi, K.: 16.7-fA/Cell Tunnel-Leakage-Suppressed 16-Mb SRAM for Handling Cosmic-Ray-Induced Multierrors, *IEEE Journal on Solid State Circuits*, Vol.38, No.11, pp.1952–1957 (2003).
14) Pflanz, M., Walther, K., Galke, C. and Vierhaus, H.T.: On-Line Error Detection and Correction in Storage Elements with Cross-Parity Check, *Proc. 8th International On-Line Testing Workshop*, pp.69–73 (2002).
15) Shin, J.L., Petrick, B., Shingh, M. and Leon, A.S.: Design and Implementation of an Embedded 512-KB Level-2 Cache Subsystem, *IEEE Journal on Solid State Circuits*, Vol.40, No.9, pp.1815–1820 (2005).
16) Shivakumar, P. and Jouppi, N.P.: CACTI 3.0: An Integrated Cache Timing, Power, and Area Model, Technical Report WRL-2001-2 HP Labs Technical Reports (2001).
17) Skadron, K. and Clark, D.W.: Design Issues and Tradeoffs for Write Buffers, *Proc. IEEE International Symposium on High-Performance Computer Architecture*, pp.144–155 (1997).
18) Suzuki, T., Yamagami, Y., Shibayama, A., Akamatsu, H. and Yamauchi, H.: A Sub-0.5-V Operating Embedded SRAM Featuring a Multi-Bit-Error-Immune Hidden-ECC Scheme, *IEEE Journal on Solid State Circuits*, Vol.41, No.1, pp.152–160 (2006).
19) Wendell, D., Lin, J., Kaushik, P., Seshadri, S., Wang, A., Sundaraman, V., Wang, P., McIntyre, H., Kim, S., et al.: A 4 MB On-Chip L2 Cache for a 90 nm 1.6 GHz 64 b SPARC Microprocessor, *Proc. IEEE International Symposium on Solid-State Circuits Conference*, pp.66 (2004).
20) Zhang, W.: Replication cache: A small fully associative cache to improve data cache reliability, *IEEE Trans. Comput.*, Vol.54, No.12, pp.1547–1555 (2005).
21) Zhang, W., Gurumurthi, S., Kandemir, M. and Sivasubramaniam A.: ICR: In-Cache Replication for Enhancing Data Cache Reliability, *Proc. International Conference on Dependable Systems and Networks* (*DSN-2003*), pp.291–300 (2003).

(Received May 8, 2006)
(Accepted August 13, 2006)

Luong Dinh Hung is currently a Ph.D. student in Information and Communication Engineering in The University of Tokyo. He received the M.E. degree in Information and Communication Engineering from The University of Tokyo in 2004. His research interests are in microprocessor architecture and circuit techniques for low-power consumption and soft-error tolerance.

Masahiro Goshima was born in 1968. He received his M.E. in engineering and Ph.D. in informatics from Kyoto University in 1994 and 2004, respectively. He was a research fellow of the Japan Society for the Promotion of Science from 1994. From 1996, he was an assistant professor in the Graduate School of Informatics, Kyoto University. Since 2005, he has been an associate professor in the Graduate School of Information Science and Technology, the University of Tokyo. He has been engaging in the research area of computer architecture. He received IPSJ Yamashita SIG Research Award and IPSJ Best Paper Award in 2001 and 2002, respectively. He is a member of IPSJ and IEEE.

Shuichi Sakai received the B.S., M.E. degrees and D.E. from The University of Tokyo in 1981, 1983 and 1986 respectively. He worked in Electrotechnical Laboratory Japan from 1986 to 1990. From 1991 to 1992, he became a visiting scientist in Laboratory for Computer Science, Massachusetts Institute of Technology. From 1993 to 1996, he was a chief at Massively Parallel Architecture Laboratory in Real World Computing Partnership. He became an Associate Professor in University of Tsukuba in 1996 and came to The University of Tokyo in 1998 as an Associate Professor. From 2001, he has been a Professor in Graduate School of Information Science and Technology in The University of Tokyo. His research interests include dependable computer systems, microprocessor architecture, compiler, parallel computing, and multimedia processing. He wrote several books on logic circuits and computer architecture, including "Introduction to Logic Circuits" and "Computer Architecture with Illustrated Explanation". He is a member of IPSJ, IEEE, ACM, IEICE, and JSAI.