Security Enhancement of Cloud Servers with a Redundancy-Based Fault-Tolerant Cache Structure

Hongjun Dai^a, Shulin Zhao^a, Jiutian Zhang^a, Meikang Qiu^{b,*}, Lixin Tao^b

^a*Department of Computer Science and Technology, Shandong University, China*

^b*Department of Computer Science, Pace University, New York, USA*

Abstract

The modern chip multiprocessors are vulnerable to transient faults caused by either on-purpose attacks or system mistakes, especially for those with large and multi-level caches in cloud servers. In this paper, we propose a modified/shared replication cache to keep a redundancy for the latest accessed and modified/shared L2 cache lines. According to the experiments based on Multi2Sim, this cache with proper size can provide considerable data reliability. In addition, the cache can reduce the average latency of memory hierarchy for error correction, with only about 20.2% of L2 cache energy cost and 2% of L2 cache silicon overhead.

Keywords: Cloud server, Security enhancement, Chip multiprocessor, Fault tolerance, Redundancy-based cache structure, Cache coherence

1. Introduction

Processors in cloud servers are usually in concentrated environments with “24 hour/7 day” continuous operation, therefore their chips are more prone to soft errors [1], either caused by on-purpose attacks or system faults. In the multi-core times, these *chip multiprocessor* (CMP) architectures are more

*Corresponding author

Email addresses: dahogn@sdu.edu.cn (Hongjun Dai), zhaoshulin.cn@gmail.com (Shulin Zhao), zhangjiutian@ict.ac.cn (Jiutian Zhang), mqiupace.edu (Meikang Qiu), ltiao@pace.edu (Lixin Tao)

susceptible to transient faults **with the reasons as** the continuous reduction of supply voltages, and the shrinking of the minimal feature size [2].

In CMPs, more than 60% of chip areas are occupied by the different levels of caches, and they are more likely to be exposed to the soft errors [3, 4]. In a
10 chip, multi-level caches become larger with more complex mechanism to keep data available and fulfill data exchange speed, and they easily suffer from *multi-bit soft errors* (MBSE) [5, 6]. Many effective methods have been proposed to improve cache reliability in recent years, such as multi-bit *error correcting codes* (ECC) [7], and some redundancy-based schemes [8, 9] to enhance the
15 security of the whole system. **However, these methods have been usually used in the traditional single-core processors.** This paper aims at using a specialized cache structure to give redundancy for MBSE correction into multi-level caches, to improve the security and reliability of CMPs in cloud servers. To achieve this, we propose an additional *modified/shared replication* (MSR) cache to keep
20 recently accessed L2 cache lines copy. Once error happens, this cache can be used as the source of recovery.

Today, it is natural to use cache coherency protocols for data consistency in these CMP cores [10], such as *modified-exclusive-shared-invalid* (MESI) [11] and *modified-owned-exclusive-shared-invalid* (MOESI) [12]. MESI has defined four
25 states: *M* (modified, dirty), *E* (exclusive, clean), *S* (shared, clean), *I* (invalid). Then, an additional state *O* (owned, dirty or clean, both modified and shared) is proposed in MOESI protocol. If an *M* cache line is hit by a read-request from other processors, its state is changed to *O*, and this avoids the need to write a dirty cache line back to main memory when other processor tries to
30 read. **But, it is possible that an *S* cache line is dirty, if one of its copies in other cores is in state *O*** [13]. That means that an *S* cache line has no redundancy in main memory either. Even if copies in other processors can be used for error correction, this also may cause performance and communication bottleneck in a chip.

35 **Traditionally, some researches have improved the cache structure** to enhance L2 cache MBSE correction in the single-core processors. They put the emphasis

on the redundancy mining of L2 cache lines. In [14, 8], they have developed a low-cost mechanism to improve the reliability of the L2 caches against MBSE by increasing of “L1 to L2” and “L2 to main memory” redundancy, with an average
40 MBSE coverage of about 96%. In [4], a replication cache can be kept small while providing replicas for a significant fraction of read hits in L1, which can be used to enhance data integrity against soft errors. In [9], a *dirty replication* (DR) cache for defect tolerance uses selectively multi-bit ECC accompanied by a content addressable memory, which can search the input data in a table of
45 the stored data, and return the matching address [15]. It proposes that soft errors in L2 cache can be corrected with the redundancy data in the current cache which keeps recent dirty block copies, or in the main memory which keeps clean block copies. However, these schemes put their focus on the single-core processors only.

50 In this paper, we proposed a novel MSR cache design for **CMPs**, especially for those in cloud servers to enhance the security of the systems. MSR cache is used at L2 cache level to give extra data redundancy. Based on MESI-like protocols, it contains most recently accessed M and S cache lines, which may have no valid copies in main memory. Furthermore, the MOESI protocol also
55 has an extension with an extra N state (no sense) to improve the write hit rate. N is used to replace I when a probe write hits from other cores to L2 cache. On the replacement, the MSR cache uses a typical LRU replacement policy. when a cache line is replaced because MSR cache is full, it needs not write back to main memory, and this can reduce the main memory writes and reduce memory
60 hierarchy latency. Similar to DR cache [9], soft errors in L2 cache blocks could also be corrected with the redundancy data in MSR cache or main memory, but MSR cache has less memory accesses, lower power consumption and smaller silicon area overhead in CMPs.

In the experiments and performance evaluation, the experiments are con-
65 ducted on an improved simulator from Multi2Sim [16], and use 5 benchmarks from SPLASH-2 [17] and 4 benchmarks from PARSEC v2.1 [18] multi-thread benchmarks. According to the results, MSR cache can ensure L2 cache’s soft

error tolerance with about 20% of average M hit rate, and it also gives S cache lines redundancy for L1 caches corresponding to the L2 cache with about 40% of average S hit rate. This also shows that the MSR cache can ensure the MBSE tolerance of L2 cache and reduce the number of accesses to main memory caused by MBSE. Typically, an 8KB MSR cache can achieve more than 95% *average effective occupy rate* (AEOR) with 20.2% of the L2 cache power consumption and 2.0% of the L2 cache silicon overhead.

The reminder of this paper is organized as follows. In Section 2, the necessary background about multiprocessor cache architecture and cache reliability is introduced. In Section 3, the details of MSR cache structure are proposed, including the extended cache protocol coherency, the correcting process, and its hardware implementation. In Section 4, the experiments and the results are given to demonstrate the effects and benefits of the solution. Finally, the paper is summarized in Section 5.

2. Background and Motivation

2.1. Typical Cache Structure in CMPs

A cache coherence protocol refers to the protocol which maintains the consistency among all the caches in a system of distributed shared memory [10].

A designer of a coherence protocol must choose the states, transitions between states, and events which cause transitions. For example, Intel Core i3 Clarkdale has two cores and one 4MB common L3 cache. In this paper, the L3 cache is used both in 8-core and in 16-core *Symmetric Multi-Processing* (SMP) models for the following experiments. Generally, L2 caches in CMPs suffer more MBSE compared with those in single-core processors. That is because cache coherence is necessary to keep the consistency of shared resource data that ends up stored in multiple local caches, leading to mass inter-core communications. Take MOESI in a typical AMD64 architecture [12] as an example, which is shown in the figure 1.

Figure 1: State transition diagram of a typical MOESI coherence protocol

- *M* (Modified)-a cache line holds the most recent correct copy of data, but it is dirty and does not have a copy in other caches;
- *O* (Owned)-a cache line holds the most recent correct copy of data with copies in other caches, but only one processor can hold the data in this state and it might be dirty;
- *E* (Exclusive)-a cache line holds the most recent correct copy of data, and it is clean without a copy in other caches;
- *S* (Shared)-a cache line holds the most recent correct copy of data with copies in other caches, and is clean if do not have any copy in the *O*;
- *I* (Invalid)-a cache line does not hold a valid copy of the data, which might

be either in main memory or caches in other cores.

Then, a read or write probe request occurs when an external bus master (i.e. cache from other processor cores) needs to access the corresponding address but missed. In particular, one of the MOESI implementation in Multi2Sim uses the following protocol functions: LOAD function (first-level cache/memory only),
110 STORE function (first-level cache/memory only), FIND_AND_LOCK function (whether hit or miss, lock on if down-up access hits), INVALIDATE function, EVICT function (write back while replacement), READ_REQUEST function (up-down or down-up), and WRITE_REQUEST function (up-down or down-up).
115 In the implementation of AMD, the “down-up” read or write requests are treated with the same way as read or write bus-master probes that come from lower level, which indicate that other processors (described as “external bus masters” [16]) are requesting the data for read or write purposes.

It can be found that a vulnerable phase of cache blocks is the lifetime phases
120 of blocks in which the modified cache block has no valid copy in the other lower or same level caches in the memory hierarchy. There are some invulnerable phases, while a cache block is in M state. Regarding transitions between old and new states of all cache coherence protocols, local reads or writes on a modified cache block causes a transition from current state M to itself. Using O state
125 in MOESI causes the increased amount of shared data in same level caches and reduced amount of time a block held in the vulnerable dirty state. This makes it possible to keep sharing for dirty data items, when at least one valid copy of a cache block exists in the same level of caches, the correction ability of errors can be guaranteed by fetching the data from same level caches.

130 2.2. Cache Reliability in Processors

An easy way to improve the reliability of L2 cache is to increase its redundancy, and it has been proposed on the single-core processor architecture. In [14, 8], simple *error detection codes* (EDC), such as Hamming Distance or *cyclic redundancy codes* (CRC), are used to correct MBSE with the redundant information stored in the memory hierarchy. It also presents a structure to detect
135

and duplicate the small values at the word level, then designs a new replacement policy to further explore the redundancy in the memory hierarchy. In [9], a small full-associative DR cache is used to save recent copies of cache blocks which are written back by write-allocate L1 cache, so that the reliability is ensured with the data duplication in DR cache and main memory. When DR cache is full, the *least recently used* (LRU) cache line will be replaced and written back to the next level memory. Figure 2 depicts an L2 cache structure with DR cache. Since it does not consider the cache coherency problem, and it is not suitable for CMPs.

Figure 2: The basic organization of DR cache

2.3. Motivation

Since an S cache line often exists in multiple L1 or L2 caches at the same time, its possibility against soft errors is much higher than other cache lines. Therefore, similar to the redundancy-mining principles in DR cache [9], we introduce an additional MSR cache in this paper, to store most recently accessed S cache lines and M cache lines. This also can reduce main memory accesses latency and improve the efficiency of MBSE correction.

3. MSR Cache Designs to Enhance Security

3.1. Cache Structure

Usually, CMPs use common EDC/ECC to detect and correct transient errors in L2 cache. This can be extended by the addition of an MSR cache. Figure 3 depicts the key points of MSR cache. This cache mainly stores the redundancy of the most recently used M and S cache lines. It is designed as a fully associative cache, and each cache line has coherence state which is similar to L2 cache. Since MSR cache also stores copies of S cache lines, an L2 cache also needs to send new S cache lines to the corresponding MSR cache when L2 cache read requests occur.

Figure 3: The design diagram of MSR cache

When an MBSE is detected by EDC/ECC in an M or S L2 cache line, if there is a corresponding redundancy cache line in MSR cache, the error could be corrected by the re-write of this redundancy cache line to L2 cache. Because this is an effective redundancy of S L2 cache, it reduces the main memory accesses and the average access time caused by error correction.

3.2. Extension of Cache Coherence Protocols

As an L2 cache, each MSR cache line also needs a coherency state to identify whether it is valid clean redundancy cache line. When a S cache line is frequently
170 modified by different cores, the L2 cache lines with out-of-date copies will be
invalidated at the same high frequency. Thus, we also try to modify the cache
coherence protocol to improve the MSR cache write hit rate. An addition of a
new N state is used for those MSR cache lines which are no use any more but
will be modified soon. Indeed, they are not valid redundancies either. While
175 an L2 cache probe write hit occurs, the cache line copy of the corresponding
MSR cache should also be changed to N rather than I . Figure 4 depicts the
relationships with I states of an MSR cache line.

Furthermore, in MOESI, when an M cache line gets a probe read hit, it
turns into O without any change in content [12]. Since the data in an O cache
180 line is shared with S cache line in other cores, it is unnecessary to use O for
redundancies in MSR cache. Instead, an S cache line can be a copy of an S L2
cache line or an O L2 cache line. Figure 5 depicts the state transition diagram
of MSR cache.

3.3. Cache Content and Coherence Maintenance

185 From the MSR cache state transition strategy, we can conclude the basic
strategies of the content and coherence maintenance to enhance the security of
the systems:

- 1) Recently accessed M or S L2 cache lines should be held by the corre-
sponding MSR cache;
- 190 2) The state of a MSR cache line should be updated simultaneously with
the corresponding L2 cache line;
- 3) An MSR cache line should be changed to N if the corresponding L2 cache
line encountered a probe write hit, such as an invalidation request hit from other
processor cores.

195 Then, assuming both L1 caches and L2 caches are write-back and write-
allocate, a complete description of the state transition can be listed below:

Figure 4: Relationship between L2 states and MSR states

- 1) If an L1 cache writes an evicted cache line back to an L2 cache, write the cache line to the corresponding MSR cache of L2 cache, and set state to *M*;
- 2) If an L2 cache received write request changes the corresponding cache line state to *M*, write the cache line to the corresponding MSR cache, and set state to *M* as well;
- 3) If an L2 cache line is invalidated by a write request coming from the next level to L2 cache, set state of the corresponding MSR cache line to *N* if it hits;
- 4) If a read request makes the corresponding L2 cache line state changed

Figure 5: State transition diagram of MSR cache

205 to S (e.g. a read miss occurred), write the L2 cache line to the corresponding MSR cache, and set state to S ;

5) If any operation makes the corresponding L2 cache line state changed to E and the corresponding MSR cache hits, update the corresponding MSR cache line and set state to E ;

210 6) If an L2 cache replacement happens, it should discard the corresponding MSR cache line of victim L2 cache line, which means to set state to I .

If MSR cache is full and leads a write miss, use LRU as a simple replacement strategy. Usually, DR cache writes the victim LRU dirty cache line back to the next level memory once a replacement is needed. However, if MSR cache in
215 CMPs uses this dirty-write-back policy, it will bring much access latency to the entire memory hierarchy with the increased inter-core communication. Thus, if an MSR cache line is selected as a victim cache line, it should NOT write back to the next level memory, including M cache line.

3.4. MOESI Extensions with MSR Cache

220 Coincidentally, this MSR cache maintenance strategy matches fitly with MOESI protocols. The details are described as follows:

1) L2 cache read hit: when an L2 cache read hit occurs, the accessed L2 cache line state does not change. Thus, an L2 cache read hit should not cause the corresponding MSR cache write or update.

225 2) L2 cache read miss: when an L2 cache shared read miss occurs, the copy of new S cache line should be put in MSR cache with S . When an L2 cache exclusive read miss occurs, if the corresponding MSR cache line exists, its content should be updated, and its state should be changed to E .

3) L2 cache probe read hit: when an L2 cache probe read hit occurs at an
230 M or S cache line, it should be changed to O or S respectively. A copy of the new O cache line should be put in MSR cache with S . Thus, the corresponding MSR cache should keep a copy of the accessed L2 cache line and set the state S at this time.

4) L2 cache write: a write up-down request may let the corresponding no- M
235 cache line changed to E , but this state transition must wait for a write request to next level memory. This process will not bring state error because low level memories also have the copies of the E cache line. When an L2 cache write request makes the corresponding L2 cache line changed to E , the corresponding MSR cache line should also be updated and changed to E .

240 5) L2 cache probe write hit: when an L2 cache probe write hit occurs, it should be changed to I , and a copy of the new I cache line in MSR cache should be changed to N .

6) L2 cache invalid operation during the replacement: when an L2 read or write miss occurs, a selected victim L2 cache line should be changed to I . If a
245 copy of this victim L2 cache line resides in the corresponding MSR cache, it should also be changed to I .

Overall, the access of MSR cache does not cause data inconsistency, and does not need to write the victim cache line back to the next level memory. When a write or a write-back request comes to an L2 cache, its corresponding

Figure 6: Soft error detection and correction with MSR cache

MSR cache can be updated simultaneously from upper level memory. However, When a read request comes to an L2 cache, the corresponding MSR cache line may need the data recently read in this L2 cache, then the L2 cache may need to send the just read cache line to the corresponding MSR cache. But the corresponding MSR cache could also be updated simultaneously with the read request to responding L2 cache at this time. As a result, an L2 cache access does not need to wait for the completion of the corresponding MSR cache update. The MSR cache strategy will not bring extra deadlocks to memory hierarchy.

3.5. Error Correcting Process

Figure 6 depicts the soft error detection and correction process with *single-error-correcting and double-error-detecting* (SEC-DED) code and MSR cache. First, SEC-DED is used to detect errors in L2 caches. If a single-bit soft error in an L2 cache line is detected by the common ECC, simply use SEC-DED check bits to correct this error. If an MBSE in an L2 cache line is detected, check the coherency state of this cache line. If the state is *I*, it does not need to be corrected because it is no use, otherwise try to find the correct L2 cache copy in the corresponding MSR cache. If hits, write the correct copy to L2

cache, otherwise search for the redundancy cache line in other cores or main memory.

4. Experiments and Results

270 4.1. Experimental Setup

The experiments are conducted on an improved simulator from Multi2Sim [16] v2.4.1 to evaluate the effects of MSR cache. Table 1 and Table 2 present the configuration parameters in 4 different processor models. Cacti [19] is used to estimate the characteristics of L1 *instruction/data* (I/D) cache, L2 cache, and
275 MSR cache at 32nm node.

For Cacti 6.5 configuration, the L1 instruction cache uses ITRS-HP (high performance) transistors, the L1 data cache uses ITRS-LSTP (low standby power) transistors, and the L2 cache uses ITRS-LOP (low operating power) transistors. We use 4 different directory-based multiprocessor architectures (2-
280 core, 4-core, 8-core, and 16-core) and 4 different MSR cache sizes (4KB, 8KB, 16KB, and 32KB) to look for a proper MSR cache size.

4.2. Evaluation Results

Figure 7 (a) and (b), depict the average M line write hit rates and the average S line write hit rates of 4 MSR caches in 4 L2 caches of 8-core model, where the
285 L2 caches are fixed as 512KB 8-way associative. However, in order to get the proper MSR cache size, it still needs more data from the MSR cache occupying situation. Hence, we defined the MSR cache effective occupy rate as the max redundancy cache line number in MSR cache divided with the capacity of the MSR cache and further divided with associativity of the MSR cache. Figure 7
290 (c) depicts the AEOR of the selected benchmarks in 4 L2 caches' MSR caches of 8-core model with different sizes used in the experiment. Also, figure 8, figure 9, and figure 10 depict the corresponding MSR cache statistics of 16-core, 2-core and 8-core models, respectively.

Table 1: Processor core configuration parameters used in the experiments

Processor Core Configuration Parameters	Value
Fetch	Timeslice, decode width = 4
Dispatch	Timeslice, width = 4
Issue	Shared, width = 4
Commit	Shared, width = 4
Storage resources	40-entry private IQs, 20-entry private LSQs, 64-entry private ROB
Functional units and latency(total/issue)	4 Int Add (2/1), 1 Int Mult (3/1), 1 Int Div (20/19), 2 FP Add (5/5), 1 FP Mult (10/10), 1 FP Div (20/20)
Branch Target Buffer (BTB)	1024-entry, 4-way
Branch Predictor	Two-level, 8-entry history table, 1-entry level-1 predictor, 1024-entry level-2 predictor

4.3. Result Analysis

295 4.3.1. Energy and Area Overhead

As shown in table 3, comparing with a 32KB L1 cache, an MSR cache with 4KB or 8KB size has low dynamic energy and area overhead, whereas those overhead for a 32KB MSR cache are much higher than that of 32KB L1 cache. In addition, a 16KB MSR cache has higher dynamic energy overhead than a
300 32KB L1 cache.

Table 2: Memory hierarchy configuration parameters used in the experiments

Memory Hierarchy (2/4/8/16-core)	Value
L1 I-cache & D-cache	32KB 2-way, 64-byte blocks, 2 cycle latency (2/4/8/16 I-caches, 2/4/8/16 D-caches, respectively)
L2 cache	512KB, 8-way, 64-byte blocks, 10 cycle latency (2/2/4/8 L2 caches, respectively)
L3 cache	8MB, 16-way, 64-byte blocks, 50 cycle latency (0/0/2/2 L3 caches, respectively)
Main Memory	64-byte blocks, 200 cycle latency
D-TLB & I-TLB	256-entry, 4-way

Table 3: Cacti report for L1 instruction cache, L1 data cache, L2 cache and MSR cache

Structure	Size	Associativity	Access time	Cycle time	Energy	Area
L1 I-cache	32KB	2	0.41 ns	0.39 ns	0.0226 nJ	0.087 mm^2
L1 D-cache	32KB	2	0.57 ns	0.56 ns	0.0181 nJ	0.087 mm^2
L2 cache	32KB	8	1.12 ns	1.46 ns	0.0784 nJ	1.272 mm^2
MSR cache	4KB	full	0.24 ns	0.10 ns	0.0096 nJ	0.014 mm^2
MSR cache	8KB	full	0.27 ns	0.11 ns	0.0159 nJ	0.025 mm^2
MSR cache	16KB	full	0.33 ns	0.10 ns	0.0396 nJ	0.068 mm^2
MSR cache	32KB	full	0.41 ns	0.10 ns	0.0781 nJ	0.143 mm^2

4.3.2. Average M and S Hit Rate

We can find that with a larger size, MSR cache can get higher average M cache line hit rate. For example, with the 16KB MSR cache size in our 8-core or 16-core model, the bodytrack benchmark can get about 3 to 4 times of M hit rate in 8KB MSR cache (80.07% to 21.92%, 72.90% to 20.79%, respectively). However, the average S hit rate is not increased obviously when MSR cache size is increased. For all of the selected benchmarks, 8KB MSR cache used in L2 caches of the 2-core, 4-core, 8-core and 16-core model can reach nearly 20% in average M cache line hit rate except for lu (about 13.88%) and fludanimate

Figure 7: MSR cache statistics of 8-core model

310 (about 15.22%) in our 4-core model, and more than 40% in average S cache line hit rate except for fmm benchmark (about 36.24%) in our 16-core model.

4.3.3. Average Effective Occupy Rate

Figure 7 (c) and figure 8 (c), the AEOR of MSR caches is decreased when MSR cache size increases. For 16KB or 32KB MSR caches, the AEOR is the
 315 lowest for all of the benchmarks. For example, with 16KB MSR cache size, the AEOR for the water-spatial benchmark are 97.56% and 90.97% in our 8-core and 16-core model, respectively. While using 8KB MSR caches in L2 caches, all but 2 benchmarks (water-nsquared: 99.81%, water-spatial: 99.22%) in our 8-core model can get 100% in AEOR, whereas 4 benchmarks (lu: 98.54%, water-nsquared: 99.42%, water-spatial: 97.56%, swaptions: 99.42%) with 16KB MSR
 320 cache size in our 8-core model cannot get 100%. In our 2-core and 4-core models, all of the benchmarks with 8KB MSR cache size can get 100% in AEOR. In our

Figure 8: MSR cache statistics of 16-core model

16-core models, blackscholes benchmark with 16KB and 32KB MSR cache size can only get 77.25% and 76.22% respectively.

4.3.4. Overall Analysis

Combined with the statistics above, we can find that 8KB might be the ideal size for MSR caches used in the proposed 8-core architecture, as it is the most cost-effective with 2.0% of the L2 cache area overhead and 20% of the L2 cache energy overhead. Similarly, 8KB should also be the ideal size in the proposed 16-core, 2-core and 4-core architectures.

In the meantime, the MSR cache scheme tends to be more suitable for CMPs, compared with other methods such as DR cache. The authors of [9] have searched and simulated over the instruction sequences with relatively high L2 access rate for each SPEC2000 benchmarks on a single-core processor architecture. In comparison, MSR cache's area cost is higher than DR cache (only

Figure 9: M/SR cache statistics of 2-core model

0.3% of the L2 cache area), but this brings extra redundancies for shared-and-clean L2 cache lines, and does not need to write back to main memory.

5. Related Work

Caches consume a large portion of total on-chip cache size and processor
340 power [3]. It becomes critical to manage the reliability of the caches in order to
maintain the reliability of the entire processor system. In a conventional design,
single-bit soft errors can be corrected by common ECC, such as SEC-DED codes
[20, 21] used in L2 caches of multiprocessors [22]. While CMPs with multiple
L2 caches can cause more MBSEs [23], two adaptive ECC schemes have been
345 used to enhance soft error tolerance of L1 and L2 cache [24]. They focus on the
impacts of soft errors in L1 and L2 caches on iterative sparse linear solvers, which
can significantly reduce the soft error vulnerability in sparse linear solvers, and

Figure 10: M/SR cache statistics of 4-core model

cut the energy consumption by 8.5%. It is relative to a ECC-protected L2 cache, but it cannot fully protect L2 cache from MBSE. Furthermore, *two dimensional* (2D) array ECC codes are used to handle clustered soft errors [25]. Although one 2D array code word protects many cache blocks, the use of array codes incur significant energy cost and *Instructions per cycle* (IPC) degradation while a large amount of random soft errors happen [26].

Several architectural techniques have also been proposed to improve reliability of on-chip cache by using either redundancy or cache resizing. In [27], it proposes to maintains multiple copies of each data item, exploits that many applications have unused cache space resulting from small working set sizes, then it detects and corrects errors using these multiple copies. Then, two non-functional blocks in a cache line are compared to yield one functional block [28], and a salvage cache improves with a single non-functional block to repair several

others in the same line [29]. Generally, they are neither scalable nor flexible to be leveraged for protection of caches in CMPs.

6. Conclusions

In cloud servers, processors are more vulnerable to soft errors caused by either on-purpose attack or system mistakes with the continuous operations. To enhance the security of cloud servers, we proposed MSR cache to provide redundancy to correct MBSE for L2 cache in CMPs. It stores the copy of the recent accessed M and S L2 cache lines. According to the experiments, a common MSR cache can get more than 20% M cache line hit rate for most benchmarks with an average about 40%. It also has 20.2% of the L2 cache energy cost and 2.0% of the L2 cache area.

In future, some important aspects need to further investigating. Although the use of MSR cache is introduced, it still needs to improve the cache efficiency. One possible approach is to store multiple information into MSR cache lines for the further reuse. For example, it may useful to store the S L2 cache lines into MSR cache to reduce the communication pressures on data bus when error occurs.

Acknowledgements

This work has been partially supported by the project “Special Program on Independent Innovation and Achievements Transformation of Shandong Province, China (2014ZZCX03301)”. Prof. Qiu has been partially supported by the projects “National Science Foundation (No. 1457506)” and “National Science Foundation (No. 1359557)”.

References

- [1] J. Cao, K. Li, I. Stojmenovic, Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers, *IEEE Transactions on Computers* 63 (1) (2014) 45–58.

- 390 [2] Y. Wang, A. Nicolau, R. Cammarota, A. Veidenbaum, A fault tolerant self-scheduling scheme for parallel loops on shared memory systems, in: 2012 19th International Conference on High Performance Computing (HiPC), 2012, pp. 1–10.
- [3] S. Wang, J. Hu, S. Ziavras, On the characterization and optimization of on-chip cache reliability against soft errors, *IEEE Transactions on Computers* 58 (9) (2009) 1171–1184.
- 395 [4] W. Zhang, Replication cache: a small fully associative cache to improve data cache reliability, *Computers, IEEE Transactions on* 54 (12) (2005) 1547–1555.
- [5] M. Manoochehri, M. Annavaram, M. Dubois, Extremely low cost error protection with correctable parity protected cache, *IEEE Transactions on Computers* 63 (10) (2014) 2431–2444.
- 400 [6] M. Manoochehri, M. Annavaram, M. Dubois, Cppc: Correctable parity protected cache, in: *Proceedings of the 8th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 223–234.
- [7] A. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, S. Lu, Energy-efficient cache design using variable-strength error-correcting codes, in: *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 461–471.
- 405 [8] K. Bhattacharya, N. Ranganathan, S. Kim, A framework for correction of multi-bit soft errors in l2 caches based on redundancy, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17 (2) (2009) 194–206.
- 410 [9] H. Sun, N. Zheng, T. Zhang, Leveraging access locality for the efficient use of multibit error-correcting codes in l2 cache, *IEEE Transactions on Computers* 58 (10) (2009) 1297–1306.
- [10] D. Hackenberg, D. Molka, W. Nagel, Comparing cache architectures and coherency protocols on x86-64 multicore smp systems, in: *Proceedings of*
- 415

the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-42, 2009, pp. 413–422.

- [11] M. Dubois, F. Briggs, Effects of cache coherency in multiprocessors, *IEEE Transactions on Computers* C-31 (11) (1982) 1083–1099.
- 420 [12] P. Sweazey, Vlsi support for copyback caching protocols on futurebus, in: *Proceedings of the 1988 IEEE International Conference on Computer Design (ICCD)*, 1988, pp. 240–246.
- [13] M. Maghsoudloo, H. R. Zarandi, Reliability improvement in private non-uniform cache architecture using two enhanced structures for coherence protocols and replacement policies, *Microprocessors and Microsystems* 38 (6) 425 (2014) 552 – 564.
- [14] K. Bhattacharya, S. Kim, N. Ranganathan, Improving the reliability of on-chip l2 cache using redundancy, in: *Proceedings of the 25th International Conference on Computer Design (ICCD)*, 2007, pp. 224–229.
- 430 [15] M. Islam, S. Ali, Improved charge shared scheme for low-energy match line sensing in ternary content addressable memory, in: *Proceedings of the 2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 2748–2751.
- [16] R. Ubal, J. Sahuquillo, S. Petit, P. Lopez, Multi2sim: A simulation framework to evaluate multicore-multithreaded processors, in: *Proceedings of 19th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2007*, 2007, pp. 62–68. 435
- [17] S. Woo, M. Ohara, E. Torrie, J. Singh, A. Gupta, The splash-2 programs: Characterization and methodological considerations, in: *ACM SIGARCH Computer Architecture News*, Vol. 23, ACM, 1995, pp. 24–36. 440
- [18] C. Bienia, S. Kumar, J. Singh, K. Li, The parsec benchmark suite: Characterization and architectural implications, in: *Proceedings of the 17th inter-*

national conference on Parallel architectures and compilation techniques,
ACM, 2008, pp. 72–81.

- 445 [19] N. Muralimanohar, R. Balasubramonian, N. Jouppi, Cacti 6.0: A tool to
understand large caches, HP Research Report.
- [20] M. Qureshi, Z. Chishti, Operating secded-based caches at ultra-low voltage
with flair, in: Proceedings of the 43rd Annual IEEE/IFIP International
Conference on Dependable Systems and Networks (DSN), 2013, pp. 1–11.
- 450 [21] P. Lala, A single error correcting and double error detecting coding scheme
for computer memory systems, in: Proceedings of the 18th IEEE Interna-
tional Symposium on Defect and Fault Tolerance in VLSI Systems, 2003,
pp. 235–241.
- [22] L. Hung, H. Irie, M. Goshima, S. Sakai, Utilization of secded for soft error
455 and variation-induced defect tolerance in caches, in: Proceedings of the
Design, Automation Test in Europe Conference Exhibition, DATE, 2007,
pp. 1–6.
- [23] J. Kim, H. Yang, M. McCartney, M. Bhargava, K. Mai, B. Falsafi, Building
fast, dense, low-power caches using erasure-based inline multi-bit ecc, in:
460 Proceedings of the IEEE 19th Pacific Rim International Symposium on
Dependable Computing, PRDC, 2013, pp. 98–107.
- [24] K. Malkowski, P. Raghavan, M. Kandemir, Analyzing the soft error re-
silience of linear solvers on multicore multiprocessors, in: Proceedings
of the IEEE International Symposium on Parallel Distributed Processing
465 (IPDPS), 2010, pp. 1–12.
- [25] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, J. Hoe, Multi-bit error tol-
erant caches using two-dimensional error coding, in: Proceedings of the
40th Annual IEEE/ACM International Symposium on Microarchitecture,
MICRO-40, MICRO 40, IEEE Computer Society, Washington, DC, USA,
470 2007, pp. 197–209.

- [26] M. Zhu, L. Xiao, S. Li, Y. Zhang, Efficient two-dimensional error codes for multiple bit upsets mitigation in memory, in: Proceedings of the IEEE 25th International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT), 2010, pp. 129–135.
- 475 [27] A. Chakraborty, H. Homayoun, A. Khajeh, N. Dutt, A. Eltawil, F. Kurdahi, E < mc2: Less energy through multi-copy cache, in: Proceedings of the 2010 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES '10, ACM, New York, NY, USA, 2010, pp. 237–246.
- 480 [28] C.-K. Koh, W.-F. Wong, Y. Chen, H. Li, Tolerating process variations in large, set-associative caches: The buddy cache, ACM Trans. Archit. Code Optim. 6 (2) (2009) 1–34.
- [29] C.-K. Koh, W.-F. Wong, Y. Chen, H. Li, The salvage cache: A fault-tolerant cache architecture for next-generation memory technologies, in: 485 IEEE International Conference on Computer Design, 2009. ICCD 2009, 2009, pp. 268–274.