Future Generation Computer Systems 52 (2015) 147–155

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

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, NY, USA

HIGHLIGHTS

- Proposed a novel MSR cache design for multiprocessors to enhance security.
- MSR cache is used at L2 cache level to give extra data redundancy.
- MOESI protocol is proposed to improve the write hit rate.
- Soft errors in L2 cache blocks could be corrected with the redundancy data in MSR cache.

ARTICLE INFO

Article history:
Received 13 September 2014
Received in revised form
19 January 2015
Accepted 3 March 2015
Available online 20 March 2015

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

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.

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 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 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 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 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 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

* Corresponding author.
E-mail addresses: dahogn@sdu.edu.cn (H. Dai), zhaoshulin.cn@gmail.com (S. Zhao), zhangjiutian@ict.ac.cn (J. Zhang), mqiu@pace.edu (M. Qiu), ltao@pace.edu (L. Tao).

http://dx.doi.org/10.1016/j.future.2015.03.001
0167-739X/© 2015 Elsevier B.V. All rights reserved.

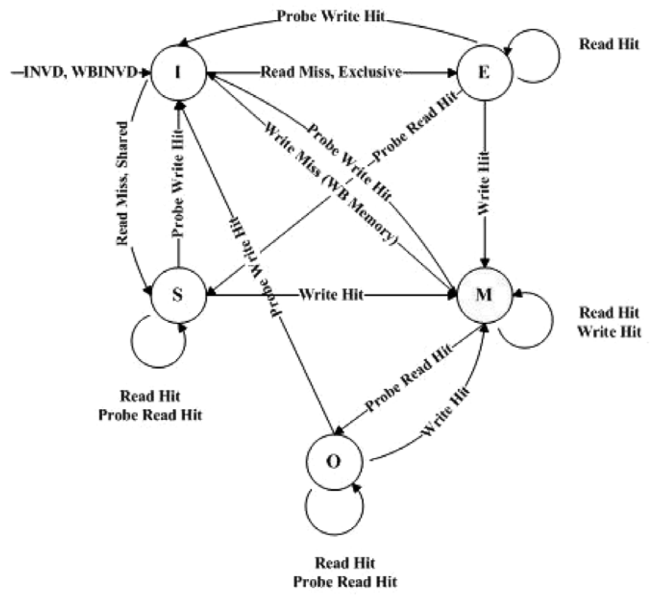

Fig. 1. State transition diagram of a typical MOESI coherence protocol.

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 processors try to 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.

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 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 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.

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 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 need not write back to main memory, and this can reduce the main memory writes and reduce memory 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 conducted 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, a 8 kB 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 4 MB 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 Fig. 1.

- *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 implementations in Multi2Sim uses the following protocol functions: LOAD function (first-level cache/memory only), 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). In the implementation of AMD, the "down–up" read or write requests are treated in the same way as read or write bus-master probes that come from lower level, which indicate that other processors (described as