Journal of Systems Architecture 000 (2016) 1–12

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

A selective protection scheme of applications using asymmetrically reliable caches

Sanem Arslan[a], Haluk Rahmi Topcuoglu[b,*], Mahmut Taylan Kandemir[c], Oguz Tosun[a]

[a] Computer Engineering Department, Bogazici University, 34342, Istanbul, Turkey
[b] Computer Engineering Department, Marmara University, 34722, Istanbul, Turkey
[c] Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802, USA

ARTICLE INFO

Article history:
Received 30 June 2016
Revised 13 December 2016
Accepted 16 December 2016
Available online xxx

Keywords:
Asymmetric cores
Selective protection
Reliability

ABSTRACT

Cache structures in a multicore system are highly vulnerable to soft errors. Enabling fault tolerance capabilities on all cache structures in a system is inefficient in terms of performance and power consumption. In this study, we propose an enhanced protection mechanism for code segments, which are critical in terms of reliability, by utilizing asymmetrically reliable cores under performance and power constraints. Our proposed system contains at least one high-reliability core, which has an ECC-protected L1 cache, and several low-reliability cores, which have no protection mechanisms. Reliability-based critical code regions are assumed to be high-priority functions, which are extracted by examining the execution time percentages and the program's call graph in our framework, statically. Software threads that invoke one of the high-priority functions are bound to the high-reliability cores dynamically during the execution, while the threads that execute the remaining functions are bound to the low-reliability cores. As part of the experimental analysis, our proposed framework is compared with traditional fully protected and unprotected configurations with respect to performance, power and reliability metrics for various applications of the benchmarks. Our framework exploits the benefits of providing the reliability-based critical regions of the applications exclusively by offering notable power and cost savings with close performance and reliability values for the set of functions reported in the experimental results.

1. Introduction

Modern architectures are vulnerable to soft errors [1] due to shrinking transistor sizes and high frequencies. Cache structures, which take large space on a chip compared with other parts, become increasingly susceptible to soft errors [1]. These types of errors might affect the target application dramatically. For instance, a single transient error on a safety-critical application such as a program that controls a nuclear power plant or a missile may result in a disaster. On the other hand, the consequence of a soft error on a molecular dynamics application with self-correcting capability (i.e., one that uses iterative solvers) may increase the execution time noticeably in spite of the fact that the application still finishes successfully [2]. For this reason, reliability might be a primary metric for some applications in hardware and software designs.

A temporary condition in a semiconductor device, which is known as a soft error, may corrupt the data stored in the memory [3]. This kind of error arbitrarily happens and may alter the data or may halt the execution of the target program. The major reasons for this type of error include alpha particles, high-energy cosmic rays and induced electrical noise from a power supply [3].

Error Correcting Code (ECC) is a commonly used technology to protect cache memories. Single Error Correction Double Error Detection (SEC-DED) technology can correct single-bit errors and detect two-bit errors. Prior research applies fault tolerance strategies unselectively to all caches, which results in inefficient performance, power consumption and cost. The goal of our study is thus to provide a reliability framework by utilizing adequate extra hardware under performance and power constraints. The main idea of our approach is to extract the code segments that are critical in terms of reliability in an application and map the application threads to the asymmetrically reliable cores according to their needs.

A chip multiprocessor framework, which contains at least one high-reliability core and several low-reliability cores, was proposed and evaluated in our recent study [4]. High-reliability cores provide ECC protection on their L1 instruction and data caches.

* Corresponding author.
E-mail addresses: sanem.arslan@boun.edu.tr (S. Arslan), haluk@marmara.edu.tr (H.R. Topcuoglu), kandemir@cse.psu.edu (M.T. Kandemir), tosuno@boun.edu.tr (O. Tosun).

http://dx.doi.org/10.1016/j.sysarc.2016.12.004
1383-7621/

Conversely, low-reliability cores do not provide any protection mechanism. In this paper, we improve our framework by extracting the reliability-based critical code regions of the applications by utilizing execution time percentages and the program's call graph, statically. We first profile the applications to determine functions with higher priority. Then, these high-priority functions are protected more conservatively than the other functions by utilizing asymmetrically reliable cores.

In this paper, a comparative study is performed to show the efficiency of our framework by using a diverse set of applications from benchmarks. We report the performance, power consumption and reliability results of different applications with our proposed framework compared with traditional caches. Our experimental evaluation shows that our proposed approach takes advantage of protecting only functions with higher priority. Our framework based on asymmetrically reliable caches presents comparable performance and reliability results with fully protected caches, while providing lower power consumption and cost values for a set of functions.

The remainder of this paper is organized as follows. We discuss related work in Section 2. We present the method used to extract the reliability-based critical regions of applications in Section 3. The properties of our approach (i.e., the architecture and execution model details) are also presented in the same section. The experimental setup used in our evaluations and results from our experimental analysis are presented in Section 4. We conclude the paper in Section 5.

2. Related work

Hardware and software-based reliability approaches are the most common studies in the literature. The physical replication of hardware units is a common method for hardware-based techniques [5]. ECC-based techniques are designed variously to provide protected cache memories. Variable-Strength Error Correcting Codes (VS–ECC) [6] are examples of where ECC strength is changed in diverse cache lines by online testing. In another study, Hi-ECC provides protection at coarse granularity by decreasing the cost of strong ECC [7]. Virtualized ECC [8] provides flexible memory protection by storing the correcting part of the check codes in the main memory. In the case of an error, their approach re-fetches the additional correction resource.

Software-based techniques provide solutions at the software-level by adding instructions into the initial program. Information, data and time redundancy are common methods used at this level. Zhao et al. [9] provide adaptive replication to maintain data reliability in shared multicore caches. In several studies, operating system data structures are protected by software-based fault tolerance [10], and checkpoint recovery [11].

On the other hand, cross-layer approaches which offer architecture- and application-level solutions to the reliability problem are presented in the literature [12]. Hardware faults are recovered at the software-level in Relax [13], where vulnerable code regions are extracted. When a fault occurs, these regions are repaired by either re-executing or ignoring the output of that region. Vulnerable code regions are selected based on discarding the calculations in the case of a fault. Sampson et al. [14] offer a programming model, named EnerJ, in which the user classifies application data as approximate or critical. In this study, calculations on approximate data are ensured by using low-reliable, low-energy hardware components; and computations on critical data are ensured by using high-reliable, high-energy components. A framework that categorizes application data as critical or non-critical is proposed by Arslan et al. [15]. They offer a protection mechanism that utilizes asymmetrically reliable cores to protect critical data.

In the programming model proposed by Carbin et al. [16], the user can figure out reliability issues at the application-level. They provide probabilistic hardware models to perform these requirements, executing on unreliable hardware. Chisel [17], in another study, picks the reliable operations and data automatically. Leem et al. [18] propose an error resilient system architecture which provides a reliable processor to manage algorithmic flow and a set of unreliable processors to execute slave tasks. In this study, application source code has to be rewritten and the usage of reliable core is effective by utilizing many parallel threads. In another study, Rehman et al. [19] provide a system that chooses a task among diverse alternatives and utilizes error recovery cores asymmetrically. Low-robust tasks are bound to the reliable cores and high-robust tasks are bound to the unreliable cores. They also provide a compiler-based solution that changes the software to decrease the vulnerability of critical instructions on unprotected hardware. A protection framework for inter-thread communications is proposed by Yetim et al. [20]. Their system converts possible fatal communication errors into data errors by utilizing application-level information. A model-driven framework for fault-tolerant embedded systems is proposed in another study [21]. Their system performs design space exploration with reliability requirements by obtaining the application description and underlying platform.

Studies have also examined asymmetric multicore architectures. Suleman et al. [22] offer asymmetric CMPs to execute critical sections faster. A large core (a high-performance core) that executes critical sections to preserve the mutual exclusion property faster, while several smaller cores (low-performance cores) execute the remaining instructions. Application threads visit the large core to execute critical sections by running faster. This approach decreases the waiting time of the threads for the critical sections.

Asymmetric multicore architectures are also studied in terms of reliability. Ungsunan et al. [23] offer a multicore system that differs in fault tolerant hardware used in cores. Their goal is to categorize software applications as critical or non-critical and to run critical applications on higher-reliability cores. They determine the criticality of the applications in advance, and the applications continue running on higher-reliability cores until they finish. Luo et al. [24] offer the idea of heterogeneous-reliability memory. In this study, application data is categorized based on error tolerance and they store less vulnerable data on lowly protected memory and more vulnerable data on highly protected memory. Their work separates the memory based on application data to decrease the memory cost of data servers.

Several studies extract critical code regions in an application. Burtscher et al. [25] assume that the code regions that have higher execution time percentages are more critical than the other parts. Subotic et al. [26] claim that the code regions that could be improved with performance optimizations and those that have maximum speed up in the case of optimizations are selected as the most significant regions. Carbin et al. [27] extract the critical and forgiving regions in an application by utilizing input fuzzing methods. They give different inputs to the program and determine critical code regions depending on the program path. Duque et al. [28] propose a system that extracts the task vulnerability and criticality metrics specific to an application by examining ready task graphs, statically.

There are considerable differences between our framework and the related work presented in this section. In our framework, the high-reliability cores are not reserved for a specific process or thread. Various threads may utilize the high-reliability cores during the execution. In this way, the dynamic allocation of application threads and a scheduling technique are required in our system. Additionally, our proposed system depends on the individual cache structures of cores as opposed to external storage. On the other hand, although we adapted the methods proposed in [28] to

Please cite this article as: S. Arslan et al., A selective protection scheme of applications using asymmetrically reliable caches, Journal of Systems Architecture (2016), http://dx.doi.org/10.1016/j.sysarc.2016.12.004