Microelectronics Reliability 62 (2016) 178–190

Contents lists available at ScienceDirect

Microelectronics Reliability

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

Using instruction result locality and re-execution to mitigate silent data corruptions

Alireza Tajary, Hamid R. Zarandi *

Department of Computer Engineering and Information Technology, Amirkabir University of Technology (Tehran Polytechnic), Hafez Ave., Tehran, Iran

ARTICLE INFO

Article history:
Received 17 October 2015
Received in revised form 3 February 2016
Accepted 23 March 2016
Available online 3 April 2016

Keywords:
Silent data corruption
Error detection
Error correction
Soft error
Temporal redundancy

ABSTRACT

In this paper, a method to mitigate silent data corruptions (SDCs) is proposed. This paper, first, shows and characterizes instruction result locality based on several simulation results and next, proposes an architecture called instruction value history table (VHT) to detect SDCs. In the case of fault detection, extra instruction redundant execution is utilized to assure fault existence. If outcome of the new redundant execution is different from that of previous one, a fault occurred, otherwise the first execution will be correct. In order to correct any detected faults, third redundant execution of the instruction is performed. Having three values from three redundant instruction executions, makes the correction of the fault feasible. The main advantage of this method is to detect any error which is not detectable by traditional protection codes like parity and SEC-DED. In other words, this method detects SDCs or any multiple faults which are not detectable by protection codes. Various soft error injections have been applied on Alpha processor for several PARSEC benchmarks. Experimental results show that the method can detect up to 70% of injected SDCs.

1. Introduction

The trends in technology scaling have led to an exponential growth in the number of on-chip transistors and significant reductions in the voltage levels of a chip. However, the use of deep sub-micron VLSI technologies in the fabrication magnifies the sensitivity of processors to transient and permanent faults [1] such as electromigration [1], stress migration [1], time-dependent dielectric breakdown [1], and soft errors [2].

Soft errors have become an important factor in degrading the reliability of current high-performance processors [3–8]. They occur massively due to the electronic noises caused by energetic nuclear particles, such as alpha-particles, neutrons, and pions, from the environments [2,9,10]. Another side effect of current nano-scale technologies is the increased likelihood of single event multiple upsets (MBUs) in the processor internal registers and memory elements.

In the past, several different strategies have been investigated to avoid, detect and recover soft errors [4,5,11–17]. Depending on whether an active error is detected, corrected or not, three situations may take place: 1) the error is detected and corrected by a fault-tolerant method, 2) the error is detected, but unrecovered by a fault-tolerant method, this type is called detected unrecoverable errors (DUEs), and 3) the error is not detected and escaped from fault detectors, this type is called silent data corruptions (SDCs). SDCs are more serious than DUEs, since DUEs are detected by a mechanism. There are also several evidences that SDCs may even occur during the normal execution of a system [3,18,19].

In order to prevent, detect and correct SDCs, redundancy is used in all previous methods [12,14,16,17,20–25]. Based on the nature of SDCs, presenting a method, which is based on the behavior of system is necessary to detect those parts of SDCs which are not covered by available techniques with negligible false positive.

In this study, an architecture to detect and correct silent data corruption (SDC) errors during instruction execution cycle is proposed. The architecture gets the behavior of system operations using the two following methods: 1) instruction result history table and 2) instruction redundant execution. After detection of an error, it is corrected using third execution of instruction. The analysis of several experimental studies on different benchmarks shows that about 80% of instructions produce repeatable results. In other words, instruction results have temporal value locality. Therefore, it is possible to provide a history table that stores recent instruction results. The proposed detection method is applied using this table before committing stage of a processor pipeline. Any mismatch in comparisons shows a potential fault during instruction execution cycle. In order to assure existence of a fault (checking potential fault), this instruction would be re-executed again. If the outcome of the new execution is different from that of the previous one, the fault happens in one of the two instruction executions and then the fault is detected, otherwise the first execution is correct. In order to correct any detected faults, third execution of the instruction is performed. Having three values from three redundant instruction executions makes the correction of the fault feasible. The main advantage of this method is to detect any errors which are not detectable by traditional protection codes like parity and SEC-DED. In other words, this

* Corresponding author.

http://dx.doi.org/10.1016/j.microrel.2016.03.029
0026-2714/

method detects SDCs or any of the multiple faults which are not detectable by protection codes. Various soft error injections have been performed on Alpha processor for 11 PARSEC benchmarks. Experimental results show that the method detects up to 70% of injected SDCs with about 2% performance overhead. Here, 70% of SDCs can be detected without any care to data bit pattern checking. It has been shown that SDCs escape from traditional error detection mechanisms which are based on data bit patterns like parity and coding techniques. In other words, the proposed method can detect 70% of all undetected errors, which may escape from other previous methods. Hence, the method is orthogonal to all other error detection mechanisms like coding techniques, and can also be used along with other methods to improve whole detection coverage.

The rest of the present study is structured as follows: Section 2 reviews the existing literature. Section 3 presents the proposed method and discusses the implementation of the architecture in a baseline core. Detailed evaluation of the proposed method in terms of fault detection coverage and overheads will be presented in Section 4. Finally, a summary of findings will be provided in Section 5.

2. Related works

Several methods have been presented for tolerating transient faults in processors trying to provide perfect fault coverage [11,13–15,20,24,28,29]. These methods can be divided into three categories [14]: redundant execution [13], anomaly detection [15], and dynamic verification [11,20]. There are furthermore some speculative methods trying to provide acceptable fault coverage with low overhead [27,30].

2.1. Redundant execution-based methods

Redundant execution utilizes dual or triple module redundancy to cope with soft errors. In these methods, two copies of a thread will be executed and their results will be compared. Because soft errors are transient, the result of the second execution will be matched. These methods can be categorized as time-based redundant execution and hardware-based redundant execution. In time-based redundant execution, one process or thread is executed twice on an individual processor, however, in hardware-based redundant execution the process or thread will be executed on two individual processors. For example, Lock Stepping uses two processors (or cores) to run the same instructions; and the results of these instructions are compared before the result commitment. At least if a conflict is observed during comparison, the error will be announced. Therefore, the result of running threads will not be committed; and the corresponding instruction will be executed again. Several industrial systems utilize this method for the purpose of fault tolerance, such as mainframes [31], the Tandem S2 [32], the HP NonStop series [33], and the Boeing 777 [34]. Moreover, Virtual Lock Stepping [35] is a new version of lock stepping, which takes the advantages of virtualization to implement lock-stepping for commodity processors [35]. AR-SMT [24] and SRT [23] are two important fault-detection techniques based on redundant multithreading [23,24]. The processor model used in these methods is the SMT (Simultaneous Multi-Threading) architecture. An SMT processor has more than one thread context, thus multiple threads can be run simultaneously on the processor [23]. AR-SMT and SRT use two threads for error detection. The first one, called active thread, runs instructions, puts its results in a buffer, and continues execution. After a delay the second one, called redundant thread reaches that instructions, runs them, and checks its results with those in the buffer. If a conflict can be found in the results, there is a potential of transient error. Time overhead of SRT is less than AR-SMT, since in SRT methods there are some queues between active and redundant threads that facilitate redundant-thread execution. CRT [21] and CRTR [13] methods are another variation of SRT for multicore processors. CRT uses two cores: one for active thread and the other for redundant thread. CRTR is an enhanced version of CRT, which is enriched with error correction capability. Finally, fingerprinting [29] compares the results of instructions after a number of instructions. There is a fingerprint for each processor, and the result of each instruction changes this fingerprint. The bandwidth of comparison is lower than Lock Stepping; however, the detection latency is higher than it [29]. These methods are suitable for multicore processors, since current desktop applications cannot simultaneously exploit all available resources in multicore processors.

2.2. Dynamic verification-based methods

These approaches use dedicated hardware and software checkers to verify the validity of specific invariants in execution of threads that are supposed to be true in error-free execution [11,14,20,36]. To have high fault detection coverage, the invariants should be comprehensive. Dynamic verification was firstly introduced in dynamic implementation verification architecture (DIVA) [11]. DIVA uses a simple checker core to detect errors in a superscalar core. This method is a low-cost solution for complex superscalar processors because the checker has a small area overhead (6% for an Alpha 21264 processor [11]). However, cores in multicore processors used for server applications are simple and scalar. Therefore, the complexity of checker significantly increases the hardware overhead. Moreover, there are some methods which use instruction reuse [37] to detect fault in processors [38–40]. These methods store the operands and the outputs of instructions to verify the output of their next executions with those of operands.

2.3. Anomaly detection-based methods

These methods monitor the software behavior by low-cost hardware and software monitors to find anomalous treatment [15]. Although these methods have a low performance overhead, they impose a high fault detection latency. For example, in SWAT [14], to obtain 98% fault detection coverage, the fault detection latency is in the order of 10 billion cycles. Therefore, these methods are good for long running threads and not suitable for the threads with a low execution time. In server applications, there are some long running threads, however the execution time of request processing threads (or processes) is small and these methods cannot be used to detect fault. Therefore, the execution of thread will be finished before converting the effect of error to an anomaly treatment.

2.4. Speculative fault detection methods

Prior methods use hardware of temporal redundancy for fault detection, and therefore have a significant overhead on hardware or performance of the processor. They intend to provide perfect fault coverage in the highly specialized servers and mission critical applications that require complete protection against transient faults. However, for desktop and commodity servers, processor manufacturers try to trade-off hardware and performance costs with reliability [27]. To predict transient faults, common architectural structures in superscalar processors like branch predictor are used. With 10% performance overhead, 34% of silent data corruption faults are detected [27].The method proposed in [41] uses speculation for memory error detection. There are some methods [42,43] which use value prediction [44–47] to provide fault tolerance and reliability in processors. The method proposed in [42] uses a table with more than 1 MB capacity for fault detection. The method proposed in [43] by means of a sophisticated value predictor tries to minimize AVF based on the detection of faults for instructions with a high latency in the pipeline.

3. The proposed method

The present study is motivated by the fact that produced result for destination operand of each program instruction, has temporal locality to previous runs of this instruction. This fact is known as value locality,