Software-Implemented Hardware Fault Tolerance Experiments COTS in Space

P.P. Shirvani, N. Oh, E.J. McCluskey
Center for Reliable Computing, Stanford University

D.L. Wood
Praxis, Inc.

M.N. Lovellette, K.S. Wood
Naval Research Laboratory

Abstract

A major concern in digital electronics used in space is radiation-induced transient errors. Radiation hardening is an effective yet costly solution to this problem. Commercial off-the-shelf (COTS) components have been considered as a low-cost alternative to radiation-hardened parts. In ARGOS project[1], these two approaches are compared in an actual space experiment. We assess the effectiveness of Software-Implemented Hardware Fault Tolerance (SIHFT) techniques in enhancing the reliability of COTS.

1. Introduction

Radiation, such as alpha particles and cosmic rays, can cause transient faults in electronic systems. Such faults cause errors called *Single-Event Upsets* (SEUs). SEUs are a major cause of concern in a space environment, and have also been observed at ground level [1]. An example effect is a bit-flip — an undesired change of state in the content of a storage element. The effects in combinational circuits, e.g., an arithmetic logic unit (ALU), can lead to incorrect results.

Radiation hardening is a fault avoidance technique used for electronic components used in space. However, these components lag behind today's commercial components in terms of performance. The need for low-cost, state-of-the-art high performance computing systems in space has created a strong motivation for investigating new fault tolerance (FT) techniques. Using COTS components, as opposed to radiation-hardened components, has been suggested for building cheaper and faster systems. However, COTS components have limited FT features. SIHFT techniques provide low-cost solutions for enhancing the reliability of these systems without changing the hardware.

2. Experiment Setup

The Stanford ARGOS project [2] is an experiment that is carried out on the computing test-bed of the NRL-801: *Unconventional Stellar Aspect* (USA) experiment on the *Advanced Research and Global Observations Satellite* (ARGOS) that was launched in February 1999. The ARGOS satellite [3] has a Sun-synchronous, 834-kilometer altitude orbit with a mission life of three years. The objective of the computing test-bed in the USA experiment on ARGOS is the comparative evaluation of approaches to reliable computing in a space environment, including radiation hardening of processors. The experiment utilizes 32-bit MIPS R3000 compatible processors. The *Hard board* uses the Harris RH3000 radiation-hardened chip set, features a self-checking processor pair configuration and has *Error Detection and Correction* (EDAC) hardware for its 2MB SOI (silicon on insulator) SRAM memory. The *COTS board* uses the 3081 microprocessor from IDT and only COTS components. It has 2MB of SRAM and has no hardware error detection mechanism except for internal cache memory parity. The operating system (OS) on both boards is VxWorks. It is possible to update the software on the boards based on the results received during the mission, and test different SIHFT techniques.

This paper presents preliminary results of our experiment. We continue to collect and analyze error data.

3. Hard Board

The Hard board has hardware EDAC for memory and a self-checking processor pair. Moreover, the data and address buses have parity bits. Upon a mismatch between the master and shadow processors, an exception is generated leading to a system halt and reset. Uncorrectable memory errors or parity errors also lead to system halt.

Several errors have been observed in the Hard board. These errors have occurred during the execution of two tests: a memory test that checks for a fixed pattern in a block of memory, and a program that generates a sine table and compares it against a stored table. Four errors have occurred in the first program and three in the second. There has also been one exception that led to a system halt. For all other errors, the programs have detected the error, reported it and continued their execution. That means the master and shadow processors were in agreement on the errors. Therefore, the errors were not upsets in one of the processors. They were not cases of double errors that were not correctable by the EDAC hardware either. We may not be able to pinpoint the source of these errors but the evidence suggests that they occur in a place that is a common source for both processors. One such place is the data buffer between memory and processors.

The error rate on the Hard board is lower than on the COTS board. This discrepancy may be due to the different SRAM components on the two boards and not the different processors.

[1] This work was supported in part by BMDO/IST and administered through ONR under grant Nos. N00014-92-J-1782 and N00014-95-1-1047.

4. COTS Board

4.1. Software-Implemented EDAC

There is no hardware EDAC to protect the main memory of the COTS board. In the early stages of the experiment, we observed that SEUs corrupted the memory, forcing frequent system resets. We implemented EDAC in software and used periodic scrubbing to protect the code segments of OS and application programs (details of our technique are described in [5]). This improved the availability of the COTS board significantly. We are able to run the board continuously for more than a month with software EDAC, as opposed to a few days without software EDAC.

Hundreds of memory bit-flips have been observed by running memory tests on the COTS board. The average memory error rate calculated based on these tests and the errors corrected by software EDAC is about 5.5 upsets/MB-day. It has been observed that a single particle can affect multiple adjacent memory cells and cause *Multiple-Bit Upsets* (MBUs) [1][4]. In our experiment, MBUs constitute about 3 percent of the memory errors. Our software EDAC is designed to handle MBUs and it has successfully corrected all the cases of MBUs.

4.2. Error Detection and Recovery

Transient errors that occur in a processor can be detected by executing a program multiple times, and comparing the outputs produced by each execution. Duplication can be done at task level by the programmer or by the OS. It can also be done at instruction level during program compilation. We have developed a technique called *Error Detection by Duplicated Instructions* (EDDI) that uses the latter approach. Computation results from master and shadow instructions are compared before writing to memory. Upon mismatch, the program jumps to an error handler that will cause the program to restart. Details of this technique can be found in [6].

EDDI can only detect some of the control-flow errors. To enhance the detection coverage for this type of error, we have developed a technique called *Control-Flow Checking by Software Signatures* (CFCSS). CFCSS is an *assigned* signature method where unique signatures are associated with each block during compilation. These signatures are embedded into the program using the immediate field of instructions that use constant operands. A run-time signature is generated and compared with the embedded signatures when instructions are executed. Details of this technique can be found in [7].

To facilitate error recovery, we break a program into modules and run each module as a separate task. A main module controls the execution of all the other modules. When one of the error detection mechanisms detects an error, the erroneous module is aborted and restarted without corrupting the context of the other modules.

EDDI and CFCSS are pure software techniques for detecting hardware errors. These techniques do not require any changes in hardware or any support from the OS. We have applied the combination of EDDI and CFCSS to two sort algorithms (Insert sort and Quick sort) and an FFT algorithm. We do an assertion check after each execution to see if there was an undetected error. For sort algorithms, we check that the data is sorted correctly. For the FFT algorithm, we calculate a checksum of the results and compare it against the expected checksum that is stored in the program. So far, the programs implementing EDDI plus CFCSS have detected a total of 116 errors, there have been no undetected errors, and more than 95% of the recoveries have been successful.

5. Conclusions

The results from the Hard board shows that despite all the hardware FT techniques used in the board, there are cases of undetected errors. Even if single points of failure are eliminated by better design, additional FT techniques, perhaps in software, may still be required for high reliability.

The software-implemented error detection and recovery techniques that we have used in ARGOS have been effective for the error rate observed in the COTS board. Even though hardware EDAC would be preferable for main memory, software EDAC has provided acceptable reliability for our experiment. Our results show that COTS with SIHFT are viable techniques for low radiation environments.

References

[1] Ziegler, J.F., et al., *IBM J. Res. Develop.*, Vol. 40, No. 1, (all articles), Jan. 1996.

[2] Shirvani, P.P. and E.J. McCluskey, “Fault-Tolerant Systems in a Space Environment: The CRC ARGOS Project,” *CRC-TR 98-2*, Stanford University, Stanford, CA, Dec. 1998.

[3] Wood, K.S., et al., “The USA Experiment on the ARGOS Satellite: A Low Cost Instrument for Timing X-Ray Binaries,” Published in *EUV, X-Ray, and Gamma-Ray Instrumentation for Astronomy V*, ed. O.H. Siegmund & J.V. Vellerga, SPIE Proc., Vol. 2280, pp. 19-30, 1994.

[4] R. Reed, et al., “Heavy Ion and Proton-Induced Single Event Multiple Upset,” *IEEE Trans. Nucl. Sci.*, Vol. 44, No. 6, pp. 2224-9, July 1997.

[5] Shirvani, P.P., N. Saxena and E.J. McCluskey, “Software-Implemented EDAC Protection Against SEUs,” to appear in *IEEE Trans. on Reliability, Special Section on Fault-Tolerant VLSI Systems*, June 2000.

[6] Oh, N., P.P. Shirvani and E.J. McCluskey, “Error Detection by Duplicated Instruction in Superscalar Microprocessors,” *CRC-TR 00-5*, Stanford University, Stanford, CA, May 2000.

[7] Oh, N., P.P. Shirvani and E.J. McCluskey, “Control-Flow Checking by Software Signatures,” *CRC-TR 00-4*, Stanford University, Stanford, CA, May 2000.