Collective Pointing: Protecting Pointer ValuesAgainst Soft Errors

Collective Pointing: Protecting Pointer Values Against Soft Errors

Emrah Islek, Serdar Zafer Can and Oguz Ergin
Department of Computer Engineering
TOBB University of Economics and Technology
Ankara, Turkey
{eislek, szcan, oergin}@etu.edu.tr

***Abstract*—Soft errors are becoming a significant design concern for microprocessor designers to make reliable systems with today's manufacturing technology. Its occurrence rate (Soft Error Rate – SER) increases with lower supply voltage, higher frequency and larger integration. Reliable systems demand protection mechanisms against soft errors at their critical structures (e.g., reorder buffer, issue queue, register file etc.). In this paper, a new low-latency method is proposed to implement Error Correcting Codes to pointer value holding fields of the processor components to protect them from soft errors. Using the proposed method, pointer fields are immune to single bit errors.**

Keywords—Error correction; fault tolerance; soft errors

I. Introduction

Soft errors are emerging as an important issue in modern digital systems [1]. They are generally caused by energetic neutrons from radiation or alpha particles from packaging material. Those particles can affect sensitive regions in semiconductor devices when they hit on it [1] [2] [3] and can change a bit's state temporarily which is called Single Event Upset (SEU). When SEU occurs during a program's execution, it may cause wrong result at the end of the program, failure of the program or even a system crash.

Currently developed high tech featured reduced size technologies for manufacturing process such as 14 nm are getting more vulnerable to these transient errors. Low energy in these low scale cells to save bits cannot resist for particle strikes. Therefore, transient faults are paramount importance for new advancing microprocessor designs and it is increasing day by day with the new generation technologies [4] [5].

In the Single Event Upset (SEU) model, the single bit flip results in a new value that has a Hamming distance of one from the original value. This fact was analyzed by Biswas et al. previously for address based structures which are in fact pointer storing fields [6]. In order to protect the source and destination pointer fields, the most straight forward idea would be to replicate the actual data into all possible locations that are Hamming distance one away from the original location. This way, even if the source or destination pointer is hit by a particle the pointer still points to a location where there is the same valid data. However, the overhead of such a scheme is high since it is tough to write multiple values at once through a limited number of ports. Also these locations that are supposed to be holding the replicated values may be already occupied by other instructions which further complicate the idea. Therefore, instead of investigating an idea that requires the replication of data all around the structures we propose having all the Hamming distance one pointers point to the same location.

The rest of the paper is organized as follows: section 2 explains the proposed method – Collective Pointing – that target error avoidance in the pointer based data in the processors. Section 3 presents experimental results of the method in terms area, energy and time overhead to the system. Related work is discussed in section 4. Finally, we offer our concluding remarks in section 5.

II. Collective Pointing

In order to correct all single bit errors, which are the most common type of soft errors, we propose a mechanism that we call "Collective Pointing". In this new scheme, the processor never assigns the pointer values that are Hamming distance one away from a main pointer. Fig 1 shows the demonstration of the proposed scheme for seven bit register tags. In the baseline, all register tags point to a different register whereas in Collective Pointing, the register tags that are Hamming distance one away from a main pointer are not assigned to any instructions and point to the same location if they are encountered as a result of a fault. When selecting the main pointers there are two rules:

1) Two main pointers have to apart from each other with a hamming distance of three.

2) None of the Hamming distance one values of a main pointer should be overlapping with another main pointer's Hamming distance one value.

Collective Pointing has three issues to be solved:

1) Having multiple pointers to the same location causes inefficiency in the use of storage space and mandates the use of wider pointers.

2) The decoder block of the storage component, which the pointers are pointing, needs to be modified to allow multiple tag values to activate the same entry.

3) Pointer value allocation logic now needs to be modified to assign only main pointers.

This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant 112E004. The work is in the framework of COST ICT Action 1103 Manufacturable and Dependable Multicore Architectures at Nanoscale.

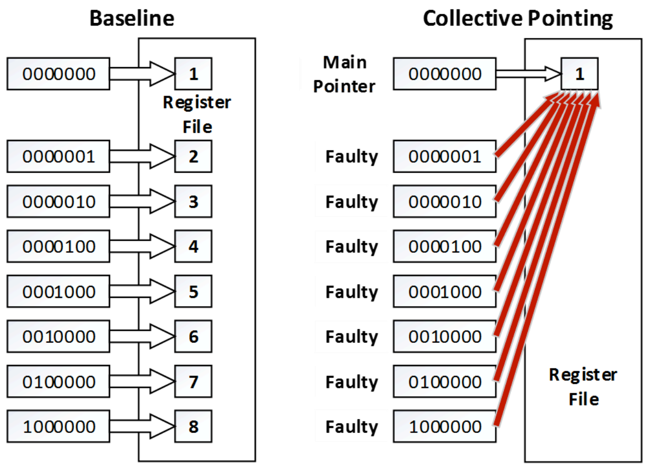

Fig 1. Demonstration of Collective Pointing

A. *The Need for Wider Tags*

Collective Pointing does need wider tags to effectively correct single bit error on each and every pointer value. This requirement is similar to the area overhead of the application of ECC where XOR trees are used to calculate the parity bits both when the value is read from and written to the storage space. In fact, the bit overhead of Collective Pointing is the same as the overhead of Hamming Code that corrects a single bit error. In order to verify this observation, we implemented a brute force main pointer generator program and obtained the numbers shown in Table I.

TABLE I. NUMBER OF LOCATIONS THAT CAN BE ADDRESSED

# Bits	# Locations at Baseline	# Locations at Collective Pointing
1	2	1
2	4	1
3	8	2
4	16	2
5	32	4
6	64	8
7	128	16
8	256	16
9	512	32
10	1024	64
11	2048	128
12	4096	256
13	8192	512
14	16384	1024
15	32768	2048
16	65536	2048

As the numbers in Table I reveal, the number of locations that can be expressed with n bits in Collective Pointing is:

$$\text{Addressable number of entries} = 2 \wedge (n - \log_2 n - 1) \quad (1)$$

This is in line with the Hamming Code's rule that states "The code word length is 2s – 1 of which s bits are the parity ones" [7]. We are in fact implementing the Hamming Code in a different fashion where instead of correcting the single bit error by using a bunch of parity bits, we make the whole value point to the same location. As it is the case with the Hamming Code, the overhead of Collective Pointing decreases with the increasing number of bits.

B. *Decoder Block*

Decoder block of a single entry in a regular SRAM bit cell array can be logically implemented with a single AND gate assuming that the inverted versions of the address bits are available. In reality, an AND gate with a high fan-in is impractical. Fig 2 shows the circuit diagram of the baseline decoder block; the output of the decoder block enables the word select driver of the entire row.

The proposed Collective Pointing scheme mandates the use of n+1 of the decoder blocks of Fig 2 in parallel for each entry and ORing their outputs, where n is the number of bits inside the tag.

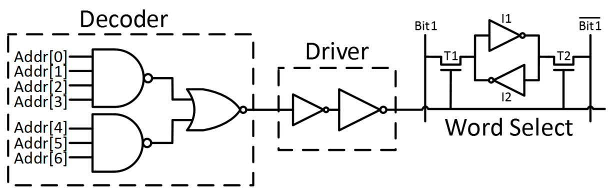

Fig 2. Baseline decoder logic of an SRAM bit cell array

In order to decrease this delay overhead we propose a different circuitry shown in Fig. 3. This circuit allows the decoding of the main pointer value faster than the faulty ones. If the outcome of the main pointer decoder is 1, the output of the multiplexer is set to V_{DD} to determine the output of the overall decoder block as 1. In this case the delay of the decoder block to be used with the proposed scheme drops just the multiplexer delay for the main pointer. The fault can be understood by monitoring the second input of the multiplexer and can be used as an error detection mechanism by the system.

C. *Pointer Assignment Logic*

Pointer value assignment logic needs to be modified in Collective Pointing. For example, the register allocation logic can be adapted to Collective Pointing by simply extending bits of physical register numbers and physical register tag fields of the related components in the processor (i.e. issue queue, reorder buffer etc.). The simplest way to achieve this is to extend pointer bits and decide which numbers will be assigned to main registers at design time. For instance, "00000000000" will take place of "0000000" and "00000000111" will take place of "0000001" and so on. This way, nothing is changed in the logic except the number of the bits.

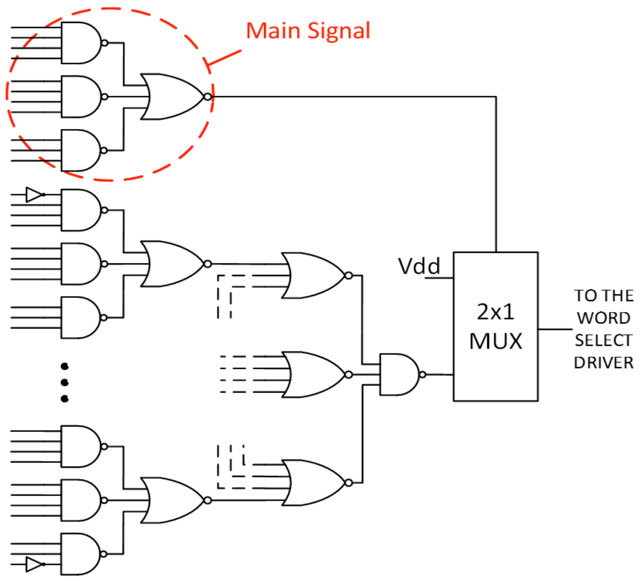

Fig 3. Proposed decoder logic for Collective Pointing

III. Experiments

The most important advantage of Collective Pointing is its low delay overhead when compared to ECC. Fig. 4 shows the hardware implementation of the ECC for 7-bit pointer values. Parity bits (P_1 P_2 P_3 P_4) are calculated before a write operation and are stored in a place together with the actual data. When reading the bits of a pointer, parity bits are recalculated (PC_1 PC_2 PC_3 PC_4) and compared with their older values to check if any fault occurs between the time interval starts from writing the address and reading it. The result of the comparison operations (C_1 C_2 C_3 C_4) should be 0 if there is no error. Single error can be corrected with ECC as can be seen from Fig. 4. ECC needs a number of XOR operations performed both during the reading and writing of the pointer value. Collective Pointing makes it possible to correct single bit errors when the delay budgets do not allow the use of ECC on the pointer tags.

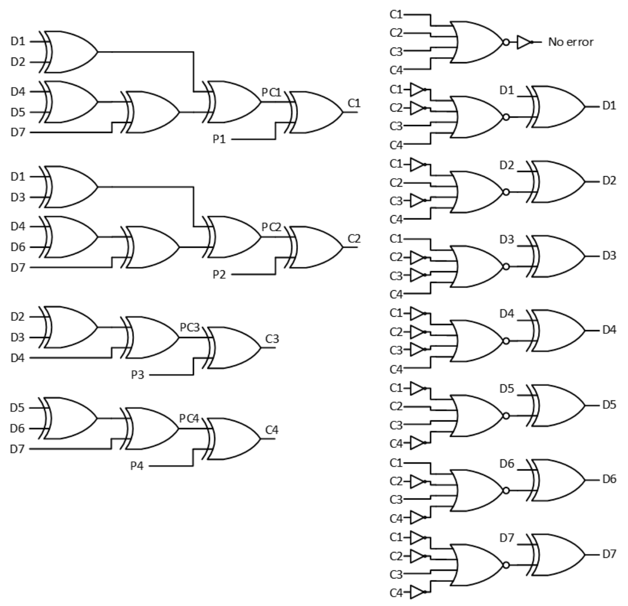

Fig 4. ECC hardware implementation

The proposed scheme also distributes the delay of error correcting circuitry across different processor pipeline stages. For example, for the register tags, the delay is distributed to the frontend, issue queue and the register file.

In order to analyze overhead levels of Collective Pointing and ECC, both of them are implemented at circuit level. Cadence design tools were used together with the 90nm CMOS (UMC) technology (V_{DD} = 1V) to compare results. ECC and Collective Pointing schemes have nearly the same area overhead as seen in Table II. However, Collective Pointing's latency and energy dissipation are much less than the latency and energy dissipation of the ECC scheme which is constructed with cascaded XOR gates. The main reason is Collective Pointing scheme is built with NAND and NOR gates which have less delay, transistor count and lower energy consumption than XOR gate. Using ECC in cycle critical structures such as register file and issue queue is impractical because of the long latency of the ECC operation. Collective Pointing enables the implementation of a SECDEC (Single Error Correction, Double Error Detection) mechanism to time critical structures due to its low-latency overhead.

TABLE II. Collective Pointing and ECC Comparison

Scheme	***Area (μm^2)***	***Energy (fJ)***	**Latency Overhead (ps)**	
			Fault	***No Fault***
Collective Pointing	444.4	109.2	176.3	104.5
ECC	469.2	387.9	358.8	

IV. Related Work

Parity checking and ECC are used to detect and correct soft errors in the data arrays of the processor [8]. These techniques rely on encoding some information from the stored data and checking this in-formation upon reading the value. Parity and ECC are widely used for cache memories but usually not on data path components due to their high encoding and decoding delay. For example, the parity protected register files of Intel's 90-nm Itanium processor needed an extra cycle to calculate parity [9]. Although reducing the latency of the ECC and parity circuits for narrow-width inputs is proposed in [10], paying the latency penalty for calculating the ECC information is unavoidable when the whole input set is considered.

Several researchers have tried to identify the effects of soft errors on the processor pipeline both at architectural level [11] [12] [13] and at gate level [14]. Effects of power saving techniques on SERs are also studied in [15]. Using time and space redundancy is a widely explored technique for detecting soft errors. In this case, either a value is replicated into more storage space and later checked with simple voting [16] [17] [18] or a value is generated multiple times with a single resource [19]. An example of time and space redundancy is the redundant multithreading proposed for error detection and correction where the same thread is replicated with some time slack and the results of both threads are later checked against each other [20] [10] [21] [22] [23] [24]. There are also some techniques to perform replication selectively when processor has idle resources in order to reduce the performance degradation caused by the concurrently running redundant threads [25] [10] [26]. It is also offered to use idle processor storage space as a repository for holding the replica values [27].

As the instructions spend more time in processor structures, their probability of getting hit by a particle increases. There were some previous efforts to reduce this vulnerable period by flushing the pipeline when instructions are stalled for a long period [2]. However, as it is the case in redundant multithreading, it is possible to observe some performance degradation. There are some other techniques in the literature that were proposed to detect soft errors, such as symptom based error detection at the hardware level [28], pure software level error detection [29] and detecting soft errors through hardware/software hybrid schemes [30].

Recently virtualizing the ECC information was proposed [31]. This way it is possible to use different coding schemes without modifying the hardware. Virtualized ECC also allows checking the information only when there is an error. Also address remapping was implemented to improve the reliability of the non-volatile memories was proposed [32]. Our proposal in this work is similar to these works in the sense that it also transfers the detection and correction mechanisms to software level and targets the memory components. However, as we have previously pointed out, regular ECC mechanisms target the errors on the actual data whereas our proposal targets the pointer values pointing to the actual data. Therefore, Collective Pointing can be used in conjunction with ECC for improved reliability.

V. Conclusion

In this work we proposed a new method that implements SECDEC mechanism to the pointer fields of the processor. With this method, pointer fields are protected from single bit upsets with lower time overhead compared with ECC. System can be aware of any fault occurrence by monitoring the second input of the multiplexer and activate any other protection mechanisms.

References

[1] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design & Test of Computers,* vol. 22, no. 3, pp. 258 - 266, 2005.

[2] C. Weaver, J. Emer, S. Mukherjee and S. Reinhardt, "Techniques to reduce the soft error rate of a high-performance microprocessor," *ISCA,* vol. 32, no. 2, p. 264, 2004.

[3] J. Ziegler et al., "IBM experiments in soft fails in computer electronics (1978–1994)," *IBM Journal of Research and Development,* vol. 40, no. 1, pp. 3 - 18, 1996.

[4] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Micro, IEEE,* vol. 25, no. 6, pp. 10 - 16, 2005.

[5] "International Technology Roadmap for Semiconductors," Semiconductors Industry Association (SIA), 2005. [Offline]. Available: http://www.itrs.net/Links/2005ITRS/Home2005.htm.

[6] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. Mukherjee and R. Rangan, "Computing architectural vulnerability factors for address-based structures," *Computer Architecture,* pp. 532 - 543, 2005.

[7] R. Micheloni, A. Marelli and R. R., "Error Correction Codes for Non-Volatile Memories", Springer, 2008.

[8] R. Phelan, "Addressing soft errors in ARM core-based designs," *White Paper, ARM,* 2003.

[9] E. Fetzer, D. Dahle, C. Little and K. Safford, "The parity protected, multithreaded register files on the 90-nm itanium microprocessor," *Solid-State Circuits, IEEE Journal of,* vol. 41, no. 1, pp. 246 - 255, 2006.

[10] Y. O. Koçberber, Y. Osmanlioglu and O. Ergin, "Exploiting narrow values for faster parity generation," *Microelectronics International,* vol. 26, no. 3, pp. 22 - 29, 2009.

[11] V. Sridharan, H. Asadi, M. Tahoori and D. Kaeli, "Reducing cache susceptibility to soft Errors," *Dependable and Secure Computing, IEEE Transactions on,* vol. 3, no. 4, pp. 353 - 364, 2006.

[12] X. Li, S. Adve, P. Bose and J. Rivers, "SoftArch: an architecture-level tool for modeling and analyzing soft errors," *Dependable Systems and Networks,* pp. 496 - 505, 2005.

[13] N. Wang, J. Quek, T. Rafacz and S. Patel, "Characterizing the effects of transient faults on a high-performance processor pipeline," *Dependable Systems and Networks,* pp. 61 - 70, 2004.

[14] K. Constantinides et al., "Assessing SEU Vulnerability," *Workshop on Architectural Reliability,* 2005.

[15] L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir and M. Irwin, "Soft error and energy consumption interactions: a data cache perspective," *Low Power Electronics and Design,* pp. 132 - 137, 2004.

[16] O. Ergin, O. Unsal, X. Vera and A. Gonzalez, "Reducing soft errors through operand width aware policies," *Dependable and Secure Computing, IEEE Transactions on,* vol. 6, no. 3, pp. 217 - 230, 2009.

[17] J. Hu, S. Wang and S. Ziavras, "In-register duplication: exploiting narrow-width value for improving register file reliability," *Dependable Systems and Networks,* pp. 281 - 290, 2006.

[18] G. Memik, M. Kandemir and O. Ozturk, "Increasing register file immunity to transient errors," *Design, Automation and Test in Europe,* vol. 1, pp. 586 - 591, 2005.

[19] M. Qureshi, O. Mutlu and Y. Patt, "Microarchitecture-based introspection: a technique for transient-fault tolerance in microprocessors," *Dependable Systems and Networks,* pp. 434 - 443, 2005.

[20] M. Gomaa, C. Scarbrough, T. Vijaykumar and I. Pomeranz, "Transient-fault recovery for chip multiprocessors," *Computer Architecture,* pp. 98 - 109, 2003.

[21] S. Mukherjee, M. Kontz and S. Reinhardt, "Detailed design and evaluation of redundant multi-threading alternatives," *Computer Architecture,* pp. 99 - 110, 2002.

[22] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading", *ISCA*, vol.28, no. 2, pp. 26 – 36, 2000.

[23] E. Rotenberg, "AR-SMT: A microarchitectural approach to fault tolerance in microprocessors," *Fault-Tolerant Computing,* pp. 84 - 91, 1999.

[24] T. Vijaykumar, I. Pomerans and K. Cheng, "Transient-fault recovery using simultaneous multithreading," *ISCA,* 2002.

[25] M. Gomaa and T. Vijaykumar, "Opportunistic transient-fault detection," *Computer Architecture,* pp. 172 - 183, 2005.

[26] J. C. Smolens, J. Kim, J. C. Hoe and B. Falsafi, "Efficient resource sharing in concurrent error detecting superscalar microarchitectures," *MICRO,* 2004.

[27] W. Zhang, S. Gurumurthi, M. Kandemir and A. Sivasubramaniam, "ICR: In-Cache Replication for enhancing data cache reliability," *DSN,* 2003.

[28] N. Wang and N. Wang, "ReStore: symptom-based soft error detection in microprocessors," *Dependable and Secure Computing, IEEE Transactions,* vol. 3, no. 3, pp. 188 - 201, 2006.

[29] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan and D. I. August, "SWIFT: Software Implemented Fault Tolerance", *CGO,* 2005.

[30] G. Reis, J. Chang, N. Vachharajani, S. Mukherjee, R. Rangan and D. August, "Design and evaluation of hybrid fault-detection systems," *Computer Architecture,* pp. 148 - 159, 2005.

[31] D. Yoon and M. Erez, "Virtualized and flexible ECC for main memory," *ACM SIGARCH Computer Architecture News - ASPLOS,* vol. 38, no. 1, pp. 397-408, 2010.

[32] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. Jouppi and M. Erez, "FREE-p: Protecting non-volatile memory against both hard and soft errors," *High Performance Computer Architecture (HPCA),* pp. 466 - 477, 2011.