Exploiting Virtual Addressing for Increasing Reliability

Yaman Cakmakci and Oguz Ergin
TOBB University of Economics and Technology
Computer Engineering Department
Email: ycakmakci@etu.edu.tr and oergin@etu.edu.tr

Abstract—A novel method to protect a system against errors resulting from soft errors occurring in the virtual address (VA) storing structures such as translation lookaside buffers (TLB), physical register file (PRF) and the program counter (PC) is proposed in this paper. The work is motivated by showing how soft errors impact the structures that store virtual page numbers (VPN). A solution is proposed by employing linear block encoding methods to be used as a virtual addressing scheme at link time. Using the encoding scheme to assign VPNs for VAs, it is shown that the system can tolerate soft errors using software with the help of the discussed decoding techniques applied to the page fault handler. The proposed solution can be used on all of the architectures using virtually indexed addressing. The main contribution of this paper is the decreasing of AVF for data TLB by 42.5%, instruction TLB by 40.3%, PC by 69.2% and PRF by 33.3%.

1 Introduction

Significant performance increase, supported by higher clock frequencies, decreasing fabrication sizes and lowering supply voltages made a set of problems called soft errors visible. Soft errors [10] are induced by alpha particles, due to decaying of material that cause alpha radiation, and high-energy neutrons and protons induced by cosmic rays. Single event upsets (SEU) may occur when these particles hit intermediate capacitive nodes of processor storage components such as SRAM bitcells and latches. Since these transient errors occur due to an incorrect charge or discharge of an intermediate capacitive node, they do not cause permanent failure in the hardware and hence are termed soft errors. With the continuous increase in chip die area [8] , it is thought that the soft error problem will increase in the future [9]. A comprehensive analysis dealing with soft-error sensitivity and pointers to methods to overcome soft errors in modern systems could be found in [1].

A useful metric for quantifying the soft error tolerance of a hardware structure is the Architectural Vulnerability Factor (AVF) [7]. Architecturally Correct Execution (ACE) analysis is used to compute AVFs of hardware structures. ACE bits are defined as bits which would have an effect on the program output when a bit flip occurs. Related to the work presented here, the AVF of address based structures were calculated by Biswas et al. [3] for data cache, data translation buffer and a store buffer.

Although compilers are primarily used as tools for maximizing the performance and minimizing the code size, previous research has shown that it is also possible to improve soft error tolerance using compiler implemented methods [4][6][5].

In this paper we propose a method to encode VPNs of a process using linear block codes and decoding them using software implemented syndrome decoders at the page fault handler for increasing the reliability of the system including TLB, PRF and the PC. The method presented can be implemented purely using software, or by working in tandem with a hardware circuit used for detecting faults.

2 Virtual Addressing Using Linear Block Codes

A VA consists of a VPN and an offset. The number of virtual pages that can be allocated depends on the page size used by the architecture. For example, the Alpha architecture uses 8K pages thus leaving room for 2^{51} pages. An important point is that current systems are not even close to using all the available pages that could potentially be allocated. It is possible to use a block encoder to encode the VPNs with the page numbers differing from each other with a hamming distance of 3, giving the chance to correct single bit errors that could occur on VA storing structures; TLBs, PRF and the PC. With the linker guaranteeing all addresses being hamming distance 3 apart at compile-time, a single bit-error in the address storing structure would not have the chance of interfering with the other addresses. Every TLB miss that is due to an unassigned virtual to physical mapping, in this case due to an error, would cause a page fault. The flow from the information source to the information sink can be adopted for a computer program in a similar

Manuscript submitted: 28-Sep-2012. Manuscript accepted: 05-Jan-2013. Final manuscript received: 15-Jan-2013

DOI 10.1109/L-CA.2013.2 1556-6056/13/$31.00 © 2013 Published by the IEEE Computer Society

vein to that used in a communications system. This process could be similarized as:

1) Information Source (Object files): The source code written in a higher language is given as an input to the compiler to be transformed into machine understandable code.
2) Transmission (Linker implemented encoding): The object files generated by the compiler are merged into a single binary by the linker. The VAs for the generated code and data segments are assigned in this stage. Similar to the encoding process used in communications systems, the addresses could be encoded in a systematic way using linear block codes so that the hamming distances between them are a minimum of three.
3) Transmission channel (Hardware structures): The transmission channel is defined as the medium which the encoded data travels through to reach the receiver. For traditional communications systems, this could be a wireless signal travelling through the atmosphere or a signal travelling through a fiber optic cable located in the ocean. In our approach, the hardware structures used to store data, such as the SRAM cells of the physical register file, are considered as the transmission medium that is prone to noise. The type of noise introduced here is due to the time spent waiting for the data causing a vulnerability to soft errors.
4) Reception: Every time an address translation is required first the TLBs are checked if the corresponding physical address resides in the related TLB. Now, let's assume a soft error has changed a single bit in the address storing structure just before this address was sent for the lookup. When the address wouldn't be found in the TLBs, this would result in a page table query. The decoder that is implemented in the page fault handler corrects the address bearing the error, and enters the corrected translation to the TLB. The address lookup process proceeds without causing any faults in the system's execution.

It can be derived from this process that each time a transient fault occurs in a virtual address storing structure, the final destination for the lookup of this address is the page fault handler (Figure-2). The errors in the examined hardware structures and their path are described below:

- PC: When a single bit error occurs in the PC, the VA stored by the PC would result in an address translation request in the end. For example, the address that was stored in the PC could be changed from $0x10000000$ to $0x10080000$. As the addresses that were encoded in link-time with hamming distance of three, when the translation for the error containing address was being made the decoder in the page fault handler can recognize that this is actually a request for $0x10000000$ and provide the physical address that corresponds to it. This way the single bit error that occured on the PC can be avoided.
- PRF: Of the three general types of instructions, branch instructions and load/store instructions are the instructions that are known to produce their results as VAs. These results reside in the physical register allocated to them until the final dependent instruction reads the result. Figure 1 shows the ratio of VA producing results to all of the instructions, which amounts to 42%. These instructions result in address lookup in the TLB as well, therefore any single bit error occuring in the physical register holding the VA value will end up in the page fault handler. The correction method is the same as described for the PC.
- TLBs: A single bit error that would affect the virtual page number part of the TLB would only cause a page fault as it is guaranteed that there would not be any hamming distance one distant address to the address that would change from a soft error as all the VPNs are encoded hamming distance three farther. This way there would not be any risk of an error occuring in the VPN storing cells responding wrongly in place for a hamming distance one farther virtual page entry.

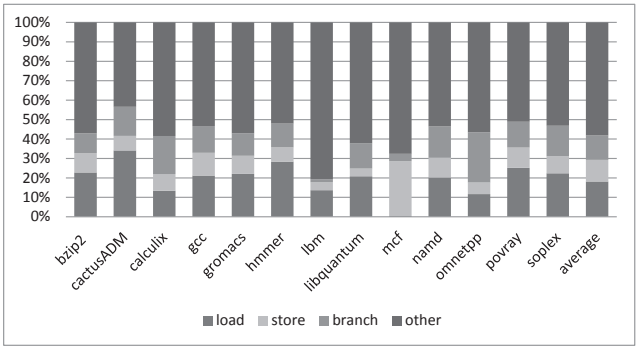

Fig. 1. Distribution of instruction types

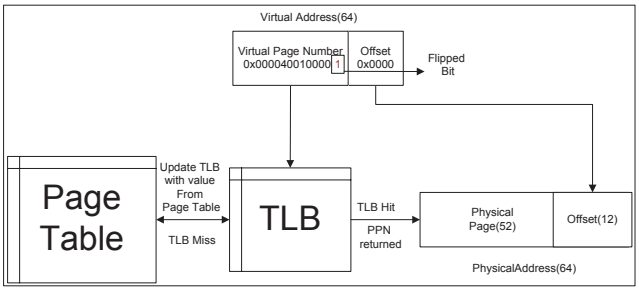

Fig. 2. Address translation

2.1 Linker Implemented Cyclic Codes

The addresses used for storing instructions and data are assigned using linkers. Cyclic codes are linear

DOI 10.1109/L-CA.2013.2 1556-6056/13/$31.00 © 2013 Published by the IEEE Computer Society

TABLE 1
Sample addressing of a program

Address	Symbol	Encoded Address
0x4001d8	__rela_iplt_start	0x200cc1d8
0x4002f8	_init	0x200cc2f8
0x4003e0	_start	0x200cc2f8
0x401690	exit	0x2017d690
0x401ae0	srand	0x2017dae0
0x4021d0	printf	0x2021f1d0
0x402450	__libc_message	0x2021f450

TABLE 2
Simulation parameters

Parameter	Configuration
Processor	Alpha 21264
Machine width	8-wide fetch,issue, commit
Window size	32 entry LSQ, 192 entry ROB, 256 entry register file
TLB	64-line data, 48-line instruction
L1 Instruction cache	32KB, 2-way set associative, 64 byte line, 1 cycle hit time
L1 Data cache	64 KB, 2-way set associative, 64 byte line, 1 cycle hit time

block codes with the additional property that c being a codeword, a cyclic shift of it is also a code word. They are a class of codes which are easy to implement. Instead of assigning the VPNs for a process sequentially, the linker can use a cyclic code encoder in order to assign page numbers. Using a $(51, 43, 3)$ encoder for this purpose is suitable for the Alpha architecture as the standard page offsets consist of 13 bits (8K pages). An example of this encoding scheme is shown in Table 1. Instead of assigning the addresses consecutively, the linker assigns the addresses as specified by the encoder, guaranteeing that the hamming distance between page numbers is at least 3.

2.2 Syndrome Decoding Inside the Page Fault Handler

Syndrome decoding is a decoding method widely used for decoding linear block codes. As the page numbers in the system are encoded as linear block codes, by incorporating this decoding technique into the page fault handler we can recover from page faults that occur due to soft errors. The above mentioned decoding method combined with our encoding method allows the system to recover from single bit flips that can alter the virtual page frame portion of the TLBs. The problem with running a syndrome decoder every time the page fault handler is run is that it results in a big performance loss. It is also possible to overcome this performance loss by employing a detection circuit that runs in parallel to the TLB which can signal whether an error is present to the operating system.When a TLB miss occurs, the detection circuit can check the virtual address that generates the TLB miss for any possible errors. In case the error detection circuit signals an error, the page fault handler is called with a notification that an error is present. The error can be signalled to the page fault handler through this new additonal hardware mechanism. In this case the software overhead at the operating system level is just transformed into a simple if clause that checks if there is an error or not. By checking the error signal when entering the page fault handler, the correction algorithm can only be run when an error is detected. Despite the performance hit of the software method, there might be cases where it might be useful such as embedded systems designs where reliability is required, but architectural modification is not possible.

3 Experimental Methodology

All the measurements regarding the microprocessor structures were made on gem5 [2] simulator by running the SPEC2006 benchmark suite on Alpha architecture. The simulator parameters are given by Table 2. The AVF measurements for the TLBs were made by assuming that only entries in the tag array that are hamming distance one apart would be tagged as ACE. PRF vulnerability has been measured by setting a timer to start the moment a physical register had its value written back. The timer would run until the last reader completes its read. Only the committed instructions were taken into account while doing this measurement. For the PC, the AVF of the structure can be considered 100% for systems that do not utilize branch prediction. For systems that employ branch prediction the squash cycles must also be considered.

4 Results

In order to assess the performance impact of the proposed error detection/correction scheme on the operating system, we implemented a syndrome decoding algorithm on Linux-3.0.0 and ran SPECint 2006 benchmarks on an x86_64 system. It can be seen from Figure 3 that introducing a purely software based block decoder in the page fault handler results in a performance degradation of around 13%. The SSE enabled version takes advantage of the population count instruction to decrease the degradation to 8% showing that it is possible to use SIMD instructions to lower the performance degradation. Besides the performance hit, the overall AVF of the translation buffers due to the linker enabled ECC technique has almost halved as shown in Figure-4. Our proposed scheme protects the virtual page part of VAs, which accounts for the reduction in the AVF.

The reduction in instruction and data translation lookaside buffers (Figure- 4) is caused due to the virtual page part being protected. For example, a TLB entry for Alpha 21264 consists of virtual tags, physical

DOI 10.1109/L-CA.2013.2 1556-6056/13/$31.00 © 2013 Published by the IEEE Computer Society

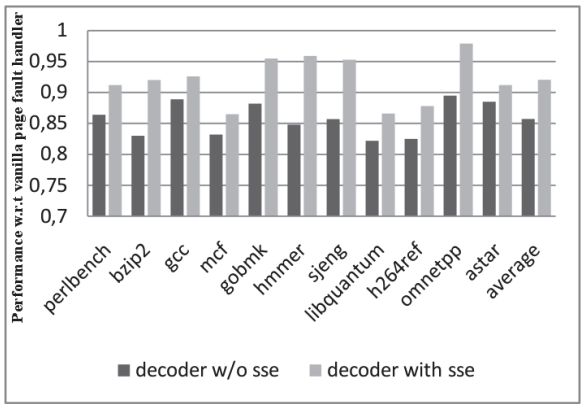

Fig. 3. SPECint performance over unmodified page fault handler

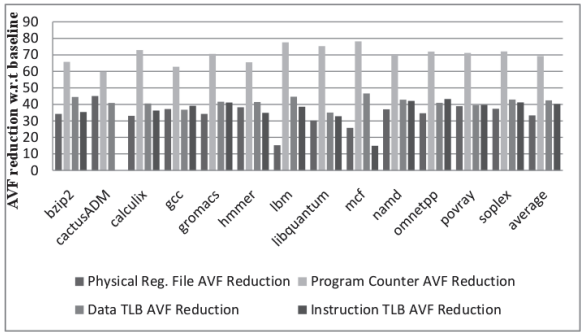

Fig. 4. Total AVF reduction for virtual address storing structures

page number, read/write permissions, address space notifier, fault on read/write, and a valid bit. The only moment a TLB hit can result in an error caused by the virtual tag is when two TLB lines with hamming-distance-one virtual tags are valid and one of them flips that different bit causing the same address to have differing translations. As superpages are being widely used in modern architectures, this method can also be applied to systems that use superpages with the cost of reducing the reliability provided, due to the increasing of the number of offset bits.

The vulnerable time for a physical register was given as the time between writeback and the reading of it by the last consumer. Load/store and branch instructions are known to writeback their results as virtual addresses. Our method can protect the PRF against soft errors that could occur when the vulnerable period for a register holding the values of load/store or branch instructions was active (Fig- 4). The reduction in PC is trivial as almost any particle strike on the PC causing a bit flip while the current instructions in the pipeline are not squashing would cause an error. Our method protects 51 bits of the PC assuming a 64 bit system using 13-bits to represent a page, reducing the AVF of the PC to $\frac{51}{64}$ (Figure- 4).

5 Conclusion

In this work, we showed a novel way of providing fault tolerance for a microprocessor system by the help of ECC techniques employed by the linker and the page fault handler of the operating system. We presented how addresses can be encoded instead of being sequentially assigned, and how they can be decoded on possible page faults due to bit flips that affect the TLBs. It has also been shown that vulnerable structures can be protected using our encoding scheme, avoiding the problems introduced by ECC encoding techniques on circuit. As a proof of concept we show that it is possible to reduce the AVF of data translation buffer by 42.5%, instruction translation buffer by 40.3%, PC by 69.2% and PRF by 33.3%.

Acknowledgements

This work was supported in part 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.

References

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

[2] Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718

[3] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. Mukherjee, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," in *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on*, june 2005, pp. 532 – 543.

[4] T. M. Jones, M. F. P. O'Boyle, and O. Ergin, in *Evaluating the effect of compiler optimizations on AVF, in Workshop on Interaction Between Compilers and Computer Architecture (INTERACT-12), 2008.*

[5] J. Lee and A. Shrivastava, "Static analysis to mitigate soft errors in register files," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '09. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2009, pp. 1367–1372. [Online]. Available: http://dl.acm.org/citation.cfm?id=1874620.1874949

[6] M. Mehrara and T. Austin, "Exploiting selective placement for low-cost memory protection," *ACM Trans. Archit. Code Optim.*, vol. 5, no. 3, pp. 14:1–14:24, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1455650.1455653

[7] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Microarchitecture, 2003. MICRO-36. Proc. 36th IEEE/ACM International Symp. on*, dec. 2003, pp. 29 – 40.

[8] D. Pham *et al.*, "The design and implementation of a first-generation cell processor," in *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, feb. 2005, pp. 184 –592 Vol. 1.

[9] SIA, in *Semiconductors Industry Association (SIA), International Technology Roadmap for Semiconductors 2005, http://www.itrs.net/Links/2005ITRS/Home2005.htm*.

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

DOI 10.1109/L-CA.2013.2 1556-6056/13/$31.00 © 2013 Published by the IEEE Computer Society