Energy-Efficient Soft Error Protection Utilizing Idle Resources

Joon Ho Kong[1], Eui-Young Chung[2] and Sung Woo Chung[3]
[1 3] Division of Computer and Communication Engineering
Korea University
Anam-Dong, Seongbuk-Gu, Seoul 163-701, Korea
[2] School of Electrical and Electronic Engineering
Yonsei University
134 Sinchon-dong, Seodaemun-gu, Seoul 120-749, Korea
E-mail: {luisfigo77, swchung}@korea.ac.kr, eychung@yonsei.ac.kr

Abstract: Soft error is a temporal malfunction, which does not leave any permanent damages in the circuits. As technology scales down, the supply voltage is decreased, leading to less critical charge. However, performance overhead and additional power consumption for soft error protection should be considered. We propose energy-efficient soft error protection scheme utilizing idle resources. In modern 4-way superscalar microprocessors, IPC is low compared to pipeline bandwidth. Thus, idle resources can be used to protect microprocessors from soft error. The ALU instructions can be duplicated in issue stage dynamically. In the proposed technique, 31.07% of total instructions is duplicated, which increases additional energy consumption by only 2.8%, on average.

1. Introduction

Soft error is a temporal malfunction, which does not leave any permanent damages in the circuits. As technology scales down, the supply voltage is decreased, leading to less critical charge. Less critical charge makes more soft errors [1]. Memory designers use error-correcting codes [3] and scrubbing (periodically reading all the cache line to detect and correct soft errors) for soft error protection [2]. In addition to the memory components, logic should be protected from soft errors: the instruction binaries are modified to duplicate all the instructions before execution [5] and the binaries are scheduled to hide performance overhead [4]. However, there is 13.3~105.9% performance overhead in 4-way superscalar microprocessors [4]. In addition to severe performance overhead, we should note there should be substantial energy overhead due to the increase execution time. As you know, energy consumption as well as soft errors should be considered for microprocessor design. Thus, there should be tradeoffs between soft error protection and energy consumption.

In this paper, we propose energy-efficient soft error protection technique that minimizes performance overhead and energy overhead. As far as we know, there has been no study on the soft errors considering energy consumption.

2. Main Idea

The key observation is that the IPC (Instruction Per Cycle) is low compared to the pipeline bandwidth in the superscalar microprocessors. Different from the previous works, instructions are duplicated in the issue stage only when the required ALU is idle and pipeline bandwidth is not fully utilized. Though the soft errors before the issue stage can not be recovered in the proposed technique, original instructions are delayed since the instructions are duplicated only when there are idle resources. Our main idea is depicted in Figure 1. In the proposed technique, the soft errors in the caches or the register file can not be recovered. However, the soft errors in the storage components can be recovered by other existing techniques such as parity bits [3] or scrubbing [3]. The only design difference in the proposed technique is the comparison logic to detect soft errors.

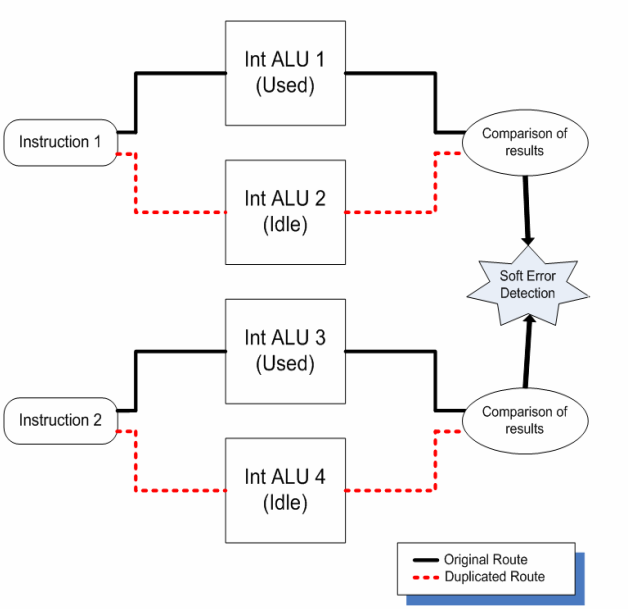

Figure 1. How to duplicate instructions

Table 1. Ratio of duplicated instructions.

Application	#Duplicated instruction / #Committed instruction (%)
gzip	28.08
vpr	34.92
gcc	33.31
mcf	24.75
crafty	30.04
parser	36.39
eon	39.01
perlbmk	27.40
vortex	27.60
twolf	26.83
bzip2	28.77
gap	35.76
Avg.	31.07

[3] Corresponding Author

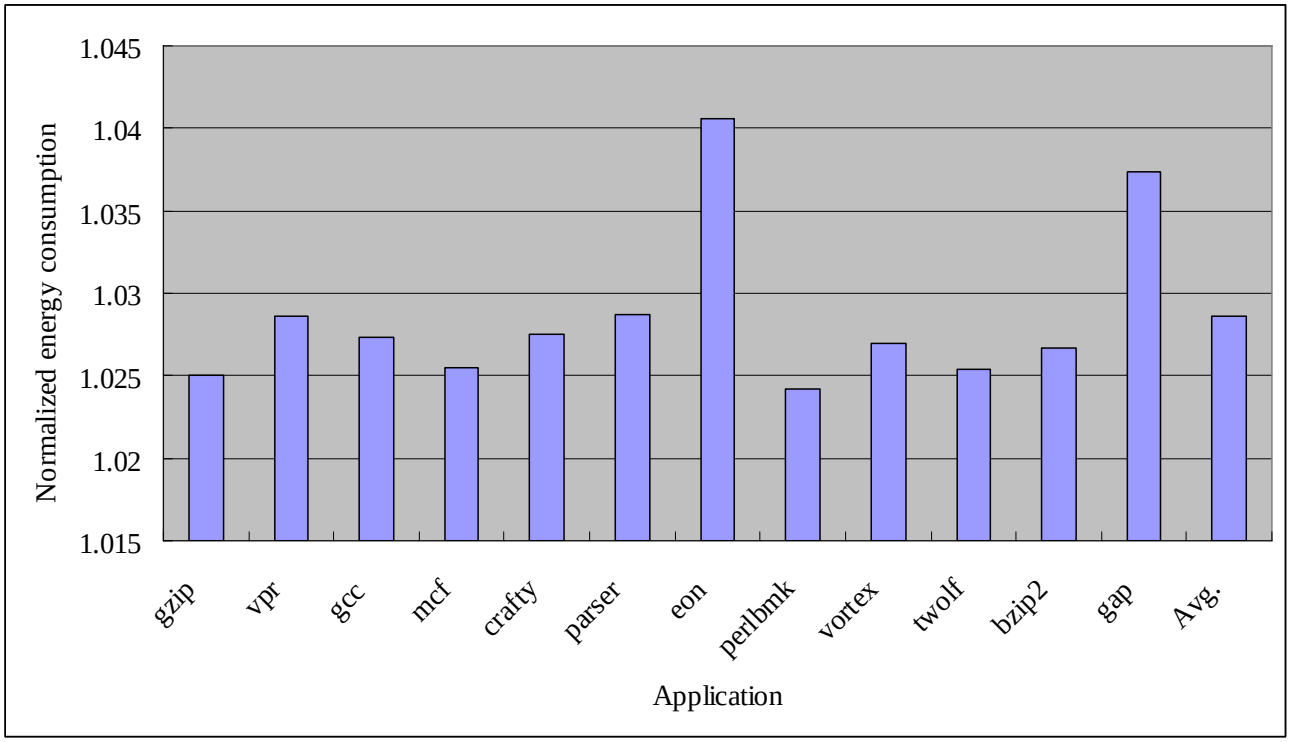

Figure 1. Normalized energy consumption with the proposed technique

3. Evaluation

To evaluate the reliability of the proposed technique where all the instructions are not duplicated for soft error protection considering energy consumption, we investigated the number of instructions that are duplicated only when there are idle resources, as shown in Table 1. On average, 31.07% of the total instructions are duplicated. We also examined extra energy consumption by using idle ALUs and pipeline bandwidth. Figure 2 shows the increased energy consumption which is only 2.8%, on average, that is much less compared to other previous techniques. In addition to negligible energy overhead incurred by the duplicated instructions themselves, there is no energy overhead caused by execution time. Note there is no increase of execution time, since only idle pipeline bandwidth and idle resources are used.

4. Conclusion

We proposed energy-efficient soft error protection technique utilizing idle resources. This technique cannot protect microprocessor perfectly, since it is limited to ALU instruction. Furthermore, all instructions cannot be duplicated, since it uses idle resources in microprocessor. However, we believe the proposed technique can be a good alternative for soft error protection in the microprocessor where energy consumption and performance overhead are critical.

5. Acknowledgement

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD, Basic Research Promotion Fund) (KRF-2006-331-D00452).

References

[1] P. Hazucha and C. Svensson, “Impact of CMOS Technology Scaling on the Atmospheric Neutron Soft Error Rate”, IEEE Trans Nuclear Science, 47(6):2586-2594, 2000.

[2] S. S. Murkerjee, J. Emer, Tryggve Fossum, and S. K. Reinhardt, “Cache Scrubbing in Microprocessors: Myth or Necessity?”, IEEE Pacific Rim Int. Symp. Dependable Computing, 37-42, 2004.

[3] V. Narayanan and Y. Xie, “Reliability Concerns in Embedded System Designs”, Computer, 39(1):118-120, 2006.

[4] N. Oh, P. Shirvani, and E. J. McCluskey, “Error Detection by Duplicated Instructions in Super-Scalar Processors”, IEEE Trans. Reliability, 51(1):63-75, 2002.

[5] M. Rebaudengo, M. S. Reorda, M. Torchiano, and M. Violante, “Soft-error Detection through Software Fault-Tolerance techniques”, Int. Symp. Defect and Fault-Tolerance in VLSI Systems, 210-218, 1999.