Transient Faults Tolerance Mechanism for Microprocessors

Toshinori Sato [1,2] Itsujiro Arita [1]
[1] *Department of Artificial Intelligence, Kyushu Institute of Technology*
[2] *Center for Microelectronic Systems, Kyushu Institute of Technology*
{tsato,arita}@ai.kyutech.ac.jp

1. Introduction

Modern microprocessors have limited hardware error detection, especially for transient faults, which are the vast majority of hardware failures[5]. Transient faults are random events, which occur when various noise sources cause an incorrect result. For example, alpha particles and other cosmic radiation can alter the state of latches and dynamic logic, resulting in logic errors. Currently, the frequency of the transient faults is low. However, small device size, increasing transistor counts, high clock frequency, and low power supply, which are accompanied with deep submicron technology, do not only reduce noise margin and reliability but also increase the impact of defects. On the other hand, portable and mobile computer platforms such as laptop and cell phone are being widely spread. For example, Java has worked on cell phones and currently several applications such as interactive video games are available. In addition, it is expected that the new phones will be used for financial service and other e-commerce businesses in the near future. For these applications, reliability and dependability are very important. From these considerations, it is expected that even low-end microprocessors should be fault-tolerant.

Current fault-tolerant techniques utilized in commercial systems such as IBM S/390 G5[5] are based on redundancies. For example, error checking is implemented by duplicating chips and comparing outputs. These techniques require two times or more hardware overhead. In addition, the duplicate and compare is adequate for only error detection. The other example is parity or error-correcting code (ECC). Contemporary microprocessors use parity for caches. However, they leave control, arithmetic, and logical functions unchecked, since it is difficult and time-consuming to check these elements[5]. Hence, low-cost fault-tolerant technique is necessary for future microprocessors.

2. Transient Faults Tolerance Mechanism

In this paper, we investigate the use of instruction reissue technique as a hardware mechanism to detect transient faults and to recover from them. Originally, the instruction reissue mechanism is proposed for incorrect data speculation recovery[2]. We modify and apply the mechanism for fault-tolerance. Since future micro-processors will utilize data speculation and include the instruction reissue mechanism, this fault-tolerant mechanism costs the least hardware overhead.

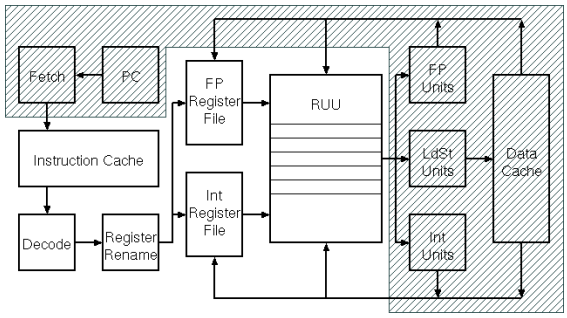

Figure 1: Protected elements

The proposed mechanism detects transient faults occurred in arithmetic and logical functions depicted as the shaded box in Figure 1. We focus on these functions because they are unchecked in modern microprocessors[5]. Instruction cache, register files, and Register Update Unit (RUU)[4] should be protected using parity or ECC, that is common for modern microprocessors. For fault detection, we propose to duplicate committed instructions and compare two results of a single instruction. The comparator should be fault free. This is possible by using large strong cells or triple modular redundancy. We use the instruction reissue to execute each committed instruction twice. When every instruction becomes ready for commitment, it is reissued in the RUU and dispatched into a functional unit again. Its execution outcome that is generated for the first time is held in the RUU and will be compared with its equivalent one when it is generated in the second time. If they do not match, a transient fault is detected.

Even when the fault detection is successful, a system with no recovery will usually hang, causing application down. The recovery can be handled by either OS or hardware. We propose two transparent hardware-based recovery schemes. One uses the existing speculation recovery mechanism for mispredicted branches, and the other is based on the instruction reissue mechanism for incorrect data speculation. In this paper, we assume the failure is transient, and thus instruction retry is successful.

That is, when a fault is detected, it is enough to re-execute the fault instruction again.

3. Evaluation Environment

We implemented a cycle-by-cycle simulator using SimpleScalar/Alpha tool set (ver.3.0a)[1]. The baseline processor model is realistic 8-way out-of-order execution superscalar processor. Dynamic instruction scheduling is based on the RUU, which has 64 entries. Each functional unit can execute any operations. The latency for execution is 1 cycle except in the cases of multiplication (4 cycles) and division (12 cycles). A non-blocking, 128KB, 32B block, 2-way set-associative L1 data cache is used for data supply. The number of its ports is 4. It has a load latency of 1 cycle after the data address is calculated and a miss latency of 6 cycles. It has a backup of an 8MB, 64B block, direct-mapped L2 cache which has a miss latency of 18 cycles for the first word plus 2 cycles for each additional word. No memory operation can execute that follows a store whose data address is unknown. A 128KB, 32B block, 2-way set-associative L1 instruction cache is used for instruction supply and also has the backup of the L2 cache which is shared with the L1 data cache. For control prediction, a 1K-entry 4-way set associative BTB, a 4K-entry gshare-type 2-level adaptive branch predictor, and an 8-entry return address stack are used. The branch predictor is updated at instruction commit stage.

The MediaBench suite is used for this study. Table 1 lists the benchmarks and the input sets. The Abbreviation column gives abbreviations for the benchmark names. Each program is executed to completion. Evaluation using SPEC2000 benchmark suit can be found in [3].

Table 1: Benchmark summary

Benchmark	Abbreviation	Input
adpcmdecode	ad-d	clinton.adpcm
adpcmencode	ad-e	clinton.pcm
epicdecode	ep-d	test.image.pgm.E
epicencode	ep-e	test_image.pgm
g721decode	g7-d	clinton.g721
g721encode	g7-e	clinton.pcm
ghostscript	gh	tiger.ps
gsmdecode	gs-d	clinton.pcm.run.gsm
gsmencode	gs-e	clinton.pcm

4. Simulation Results

This section presents our simulation results. In this paper it is assumed that we do not detect any transient fault but we only evaluate performance penalty caused by introducing the fault-tolerant mechanism. Therefore, the impact of the fault recovery mechanism on performance is not evaluated. Figure 2 shows performance loss due to duplicating all committed instructions. We use execution cycles for evaluating performance and the figure presents the percent increase of execution cycles. The performance loss is the average of 41.8% and as much as 72.3% in the case of gsmencode. This is considerably smaller than the case where the program is executed two times and the results are compared. Thus, the proposed fault-tolerant mechanism based on the instruction reissue is useful for future microprocessors with enough hardware resources.

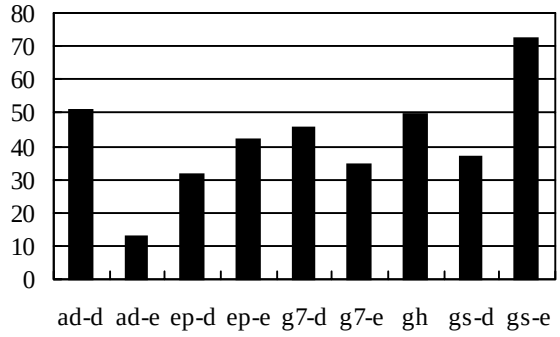

Figure 2: %Increase of execution cycles

5. Concluding Remarks

In this paper, we proposed the fault-tolerant mechanism for future microprocessors. It is based on the instruction reissue technique for incorrect data speculation recovery, and utilizes time redundancy. We evaluated our proposal using the cycle-by-cycle simulator and found that it suffered moderate performance degradation if future microprocessors would have enough hardware resources.

One of the future studies regarding the fault-tolerant mechanism for microprocessors is evaluating the impact of the transparent fault recovery mechanism on processor performance. We should construct a transient faults model for mobile environment. We should also investigate simpler fault-tolerant mechanisms than that proposed in this paper. For low-end embedded microcontrolers, it is expansive to include eight functional units

References

[1] D.Burger and T.M.Austin, “The SimpleScalar tool set, version 2.0”, ACM SIGARCH Computer Architecture News, vol.25, no.3, 1997.

[2] T.Sato, “Analyzing overhead of reissued instructions on data speculative processors”, Workshop on Performance Analysis and its Impact on Design held in conjunction with 25th Int. Symp. on Computer Architecture, 1998.

[3] T.Sato and I.Arita, “Tolerating Transient Faults in Microprocessors”, 13th Joint Symp. on Parallel Processing, 2001 (in Japanese).

[4] G.S.Sohi, “Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers,” IEEE Trans. Comput., vol.39, no.3, 1990.

[5] L.Spainhower and T.A.Gregg, “IBM S/390 parallel enterprise server G5 fault tolerance: a historical perspective”, IBM J. Res. Develop., vol.43, no.5/6, 1999.