Exploiting Replicated Checkpoints for Soft Error Detection and Correction

Fahrettin Koc, Kenan Bozdas, Burak Karsli, Oguz Ergin
Department of Computer Engineering
TOBB University of Economics and Technology
Ankara, TURKEY
{fahrettin.koc, kbozdas, ibkarsli,oergin}@etu.edu.tr

***Abstract*—Register renaming is a widely used technique to remove false dependencies in contemporary superscalar microprocessors. A register alias table (RAT) is formed to hold current locations of the values that correspond to the architectural registers. Some recently designed processors take a copy of the rename table at each branch instruction, in order to recover its contents when a misspeculation occurs. In this paper first we investigate the RAT vulnerability against transient errors. Then we analyze the vulnerability of RAT checkpoints and propose two techniques for soft error detection and correction utilizing redundantly taken copies of the entries whose content is the same with the previous and/or next checkpoints. Simulation results of the spec 2006 benchmarks reveal that on the average RAT vulnerability is 25% and checkpoint vulnerability is 6%. Results also reveal that redundancy exists at sequential checkpoint copies and can be used for error detection and correction purposes. We propose techniques that exploit this redundancy and show that faults in 41% of all checkpoints and 44% of rolled-back checkpoints can be detected and errors in 33% of the rolled-back checkpoints can be corrected. Since we exploit the already available storage, proposed error detection and correction techniques can be implemented with minimal hardware overhead.**

Keywords— Microprocessors, Register Rename, Checkpoint, RAT Vulnerability, Soft Error, Error Detection and Correction

I. INTRODUCTION

ALPHA particles released by packaging radioactive impurities and neutrons caused by cosmic particles coming from outer space are known to cause transient errors in contemporary microprocessors [8]. Capacitive nodes of the processor storage components such as SRAM bitcells and latches are the most sensitive parts to these particle hits. These hits may cause single or multi-bit transient errors since they do not cause permanent defects in the hardware and hence are called *soft errors* in the literature.

Contemporary microprocessors use performance boosting techniques like out-of-order execution, deep pipelines, and dynamic scheduling. Moreover, almost all contemporary processors use register renaming in order to cope with false data dependencies. Register renaming technique brings the use of a Register Alias Table (RAT) where all architectural to physical register mappings are stored.

Speculative execution is also another technique by taking the advantage of the information about the taken or not taken history of the conditional branches. Any speculative path with mispredicted conditional branch would need a proper roll-back mechanism for returning to a safe state just before the path. This roll-back mechanism can be constructed as a reorder buffer (ROB) based or checkpoint-based implementation.

Since errors are hazardous for correct program flow our main motivation is the identification and measurement of the most vulnerable areas against soft errors in the checkpoint-based RAT recovery methods. In this paper we start by investigating the RAT vulnerability against transient errors. The RAT contains current mappings of the architectural to physical registers and hence is important for correct execution. Then we analyze the vulnerability of the RAT checkpoints taken for recovery purposes. A checkpoint may also be crucial if it is rolled-back and written onto the RAT. Finally, we propose two techniques for soft error detection and correction utilizing redundant information stored in consecutive checkpoints. Conventional error detection and correction methods utilize redundant storage [12] with increased area overhead for the copies. However, our technique exploits already existing redundancy between consecutive RAT copies.

The rest of the paper is organized as follows: the discussion about the related work is placed in the Section II. Two main misprediction recovery approaches are detailed in section III. In section IV, vulnerability analysis on RAT and its checkpoints is given. Checkpoint assisted error detection and correction schemes and their hardware implementations are presented in section V. We present our experimental results in section VI. Finally, we offer our concluding remarks in the same section VI.

II. RELATED WORK

RAT and checkpoints can be implemented as RAM or CAM array. RAM array table is fast and scalable whereas CAM array [11] requires smaller storage area and is easy to recover. Checkpoint for a CAM array table need only copy of the valid bits whereas for a RAM array RAT whole table is copied. Random access buffer [2] checkpointing recovers in 1 clock cycle and we selected this scheme in our hardware implementation but our techniques can also be adapted to sequential access buffer scheme. Zeng et.al. [6] suggests CAM array RAT and checkpoints and storing bit vectors instead of storing whole RAT as in RAM array. However, recovery mechanism needs multiple cycles for reconstructing the rename table and hence causes roll-back performance degradation.

There are many contributions including AVF analysis like the studies of Mukherjee et. al. [13] which proposes AVF analysis on computer elements, however to the best of our knowledge the RAT and checkpoint AVF analysis are firstly introduced in this paper. Montesinos et.al. [9] investigates the register lifetime for soft error analysis on register files. The error resilience in fault tolerant systems is generally sustained with adding redundancy by copying data. As an illustration, storage-based systems use two copies for error detection and three copies for error correction where the latter is a well-known example of the triple modular redundancy mechanism [12]. However, such systems have hardware cost for extra storage and comparison circuitry. Redundant information in RAM-based sequential RAT checkpoints is the main inspiration point of our work. There are also several studies by using architectural implementations as an whole unit of error detection and correction like Sorin's contribution, an error detection unit called Argus [12]. Furthermore, there exist some well known techniques like exploiting parity bits in case of single error occurrence commonly used applied to current systems because of their high success rates. In our work, we exploit already existing redundancy in these copies of RAT for soft error detection and correction without extra hardware overhead for storage in order to build up resilient systems against multiple errors.

III. MISPREDICTION RECOVERY

Superscalar microprocessors use speculative flow control like branch target speculation to increase performance further. However, on a mispredicted branch this speculative flows need to recover to the last known safe state. This can be done by recovery mechanisms utilizing reorder buffer (ROB) or taking checkpoints of the state before branch. ROB-based mechanisms are wait, walk forward and backwards [5]. Checkpoint-based mechanisms can be implemented as sequential access (shift registers) [9] and random access [3]. The simplest way to recover from a misprediction is to keep the information for the history of the committed instructions. The table that holds the location of the commited values of the architectural registers is called the commit rename table (CRT). A scheme that makes use of the CRT to recover from branch mispredictions, starts from the beginning of the ROB and waits until the mispredicted branch commits. Once the branch is committed, contents of the CRT, which contains the last known safe state, are copied onto the RAT. Checkpoint mechanism is taking the snapshot of the RAT [14] when a conditional branch arrives to the rename stage. Recovery with checkpointing from a misprediction is a faster solution but brings higher complexity in circuit level compared to ROB-based counterparts. RAT and checkpoints can be RAM array where entries are accessed by index with architectural register ids [2], [5]. In a CAM array RAT, the number of entries equal to the number of physical registers and indexed with physical register ids [1]. In RAM array method, the whole RAT table is checkpointed, whereas in CAM array RAT only valid bits vector is needed to be copied. Checkpoints can be accessed in sequential or random access fashions [2]. Taking a snapshot of the status at each branch may create redundant states to be copied consecutively. These redundancies will be used to detect and/or correct a soft error.

IV. VULNERABILITY ANALYSIS

RAT Vulnerability: RAT contains current architectural to physical register mappings and hence is critical for the correct execution. Any error on this table may not only cause data corruption but also incorrect flow control. As an illustrative case, vulnerability of an entry in this table is directly related to the current mapping being requested by any other instruction's source or not. In other words if there is no dependent to the last mapping of an architectural register then any error on this mapping entry would be invisible in program outcome. In another case, a branch misprediction may cause a roll-back and effectively vulnerability of the table entries in the speculative path will be zero. Consequently, average AVF of each architectural to physical register mapping can be determined according to net lifetime of the mapping in a program which is the difference between total lifetime in execution and total speculative lifetime on mispredicted branch executions. The AVF of all mapping entries corresponding to each architectural register is found for AVF of the RAT.

Checkpoint Vulnerability: Considering the random access buffer checkpoint mechanism proposed in [2], M cell corresponds to typical RAT without any other specific condition and its vulnerability can be determined as in (2). However, vulnerability of the checkpoints (C) is slightly different. Vulnerability is zero for unused checkpoints since they have no effect on the program flow. Vulnerability of the rolled-back checkpoints depends on the duration starting from the taken time to the roll-back time.

The AVF of the RAT for each SPEC2006 benchmark programs are given in Fig. 2. The AVF of the RAT is minimum 14.2% for *specrand* with 32-checkpoint configuration maximum 49.4% for *leslie3d* with base configuration and 25% on the average. RAT vulnerability is the average of active life cycle of each entry in the RAT. Life

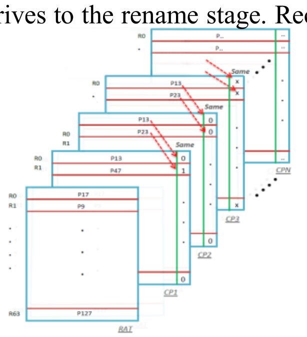


Fig. 1. Checkpoint entries with Same Bit

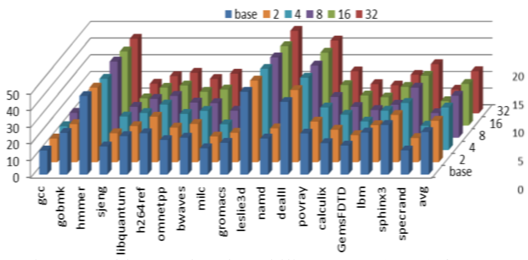


Fig. 2. Architectural Vulnerability Factor (AVF) for RAT

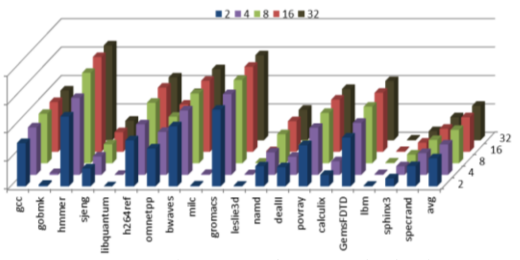


Fig. 3. AVF for RAT Checkpoints

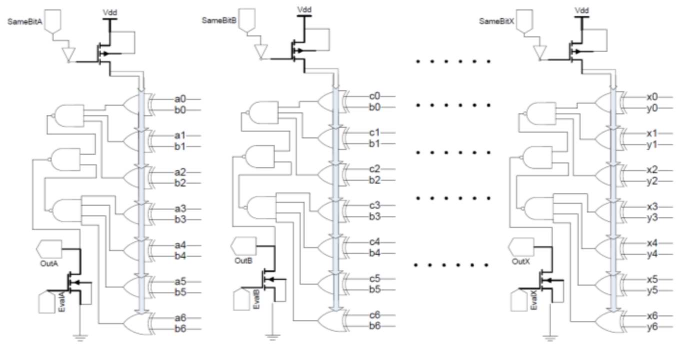

SameBitA
SameBitB
SameBitX
Vdd
a0 b0 a1 b1 a2 b2 a3 b3 a4 b4 a5 b5 a6 b6
c0 b0 c1 b1 c2 b2 c3 b3 c4 b4 c5 b5 c6 b6
x0 y0 x1 y1 x2 y2 x3 y3 x4 y4 x5 y5 x6 y6
OutA
OutB
OutX
EvalA
EvalB
EvalX

Fig. 4. 7-bit Comparator for on-the-fly check

cycle for an entry in the RAT begins with a rename and ends with the last read from the corresponding architectural register. The AVF of checkpoints for each benchmark program is given in Fig. 3. Vulnerability of the checkpoint begins at the taking a copy of RAT and ends with roll-back. Checkpoint AVF is low for some benchmarks where the number of roll-backs is small. Vulnerability increases with the number of checkpoints since more checkpoints allows more in-flight branches and decreases stalls. As a result total lifetime of the checkpoint in the storage area increases.

V. CHECKPOINT ASSISTED ERROR DETECTION AND CORRECTION

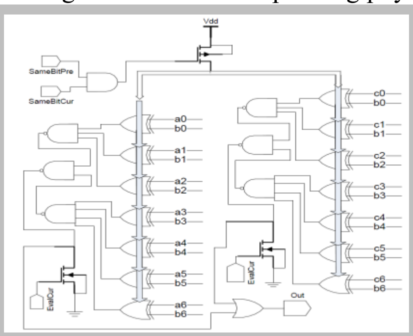
Considering an architectural register between two consecutive conditional branches, checkpoint entries in some rows (architectural to physical mappings) can be same. This situation occurs if there is no rename operation took place for the corresponding entries and corresponding physical registers are not released. An error detection algorithm may utilize this redundancy in the consecutive checkpoints. We define *comparable entries (CE)* as any individual entry in the table has a redundant copy at its neighborhoods.

We define *same* bit which guarantee that consecutive are real comparable entries. For instance, assume that two consecutive entries have the same value *X*. Having the same value does not guarantee that these two entries are comparable. In order to assure comparable entries two distinct situations must be satisfied. Firstly, there is no new rename for the corresponding register since last conditional branch point. Secondly, this same bit must act as a valid bit for each checkpoint to indicate corresponding checkpoint is not flushed due to a mispredicted older branch or its speculative branch is not committed. A conceptual view of the checkpoints with same bit is given in Fig. 1. We propose two different schemes at distinct phases for error detection in RAT, namely On-the-fly Check and On-Demand Check.

On-the-fly Check: First scheme takes place when there is a speculative flow in the program where a checkpoint is created and stored in checkpoint space. We are proposing to compare the current copy of each RAT entry with the previous copy if it is known that there is no register rename for the corresponding register since last checkpoint. We add an extra *same* bit to each entry in the table that indicates current and next copy are comparable as illustrated in Fig. 4. Each checkpoint updates these control bits in the previous checkpoint copy which will be utilized in the second scheme. The circuit is given in Fig. 4.

On Demand Check: Comparison at this scheme starts when there is a roll-back request for the corresponding checkpoint. During roll-back, each table entry is compared with the corresponding CEs in either preceding or next checkpointed tables. If *same* bit is '1' for the entry in the requested checkpoint then the corresponding entry in the next checkpoint would not be compared. This '1' indicates that there was a rename between these two neighbor checkpoints. If same bit is '0' these two entries are CEs. There are 3 possible situations. First, there are no CEs in the neighborhood of the current entry. It is not possible to detect any errors just by comparison. In the second case, there is one CE and comparison reveals error if exist. If any error is detected this could be flagged to a recovery mechanism. For the last case there are 2 CEs and error can be detected and corrected by comparing these three consecutive copies. If any error is corrected and there is no other detected error then it is safe to recover with the corresponding checkpoint. The circuit is given in Fig. 5.

Hardware Implementation: We full custom designed RAT and checkpointed tables as random access buffer scheme proposed in [2] design for UMC 90nm technology library with Cadence Virtuoso. RAT cell has eight read and four write ports as required in a 4-way machine. Checkpoint cells each have one port for read and write as only one checkpoint can be taken or rolled-back at a time. Each entry in RAT cell has seven bits for mapping 128 physical registers. Checkpointed table cells each have one read/write port. Each entry in checkpoint cell is eight bits where one *same* bit is added to indicate that current cell and the next cell are comparable entries at the corresponding rows. Roll-back and branch resolution events


Vdd
SameBitPre
SameBitCur
a0 b0 a1 b1 a2 b2 a3 b3 a4 b4 a5 b5 a6 b6
c0 b0 c1 b1 c2 b2 c3 b3 c4 b4 c5 b5 c6 b6
EvalCur
Out

a) 14-bit Comparator for on-the-demand check

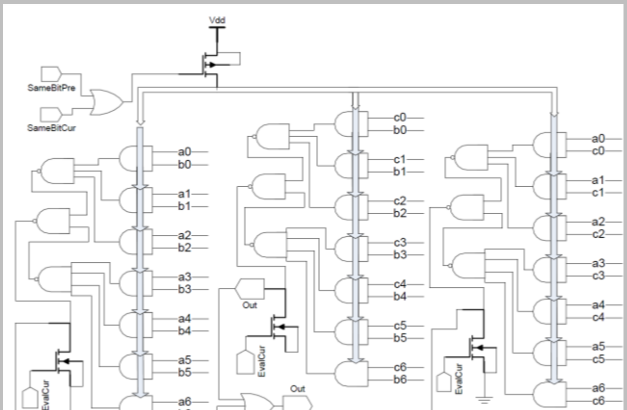

Vdd
SameBitPre
SameBitCur
a0 b0 a1 b1 a2 b2 a3 b3 a4 b4 a5 b5 a6 b6
c0 b0 c1 b1 c2 b2 c3 b3 c4 b4 c5 b5 c6 b6
a0 c0 a1 c1 a2 c2 a3 c3 a4 c4 a5 c5 a6 c6
Out
EvalCur

Fig. 5. b) Error correction circuit for on-the-demand scheme indicating that current cell has a valid copy of the RAT for an unresolved conditional branch.

The *area* of the 64x7 RAT checkpoint SRAM with one read/write port is 24,023 μm^2 and 25,492 μm^2 for a 64x8 checkpoint table. Adding a *same* bit to each entry in table for error analysis purpose brings 6% area overhead. Adding comparison circuit for on-the-fly scheme brings 5857,95 μm^2 and for on-demand scheme brings 10406.4 μm^2 extra area. Critical path *delay* is 208 ps for the baseline checkpoint table whereas it is 217 ps with the *same* bit added. Adding one bit makes critical path longer and brings 4% delay overhead. Adding an extra bit for error analysis will not alter precharge, decoder, sense amplifier, and write drive *energy* consumptions. Only word select energy will slightly increases since getting longer. Word select energy for the baseline table is 95.56 fJ and for the proposed table is 96.91fJ.

VI. RESULTS AND DISCUSSIONS & CONCLUSION

The percentage of the CEs in on-the-fly checking scheme is given in Fig. 6. The number of CEs is almost constant with the number of checkpoints for each benchmark since it depends on only the program flow. Percentage is found by averaging on the number of total checkpoints. The percentages of one CE and two CEs for on-demand scheme are given in Fig. 7 respectively. Benchmark simulation results reveal that minimum 30.22%, maximum 53.41%, and average 44.82% of the errors can be detected for the on-demand scheme. Moreover, minimum 0.42% (excluding 2 checkpoints since not applicable), maximum 49.55%, and average 33.38% of the errors can be corrected for on-demand scheme. Percentages are found by averaging on the number of total roll-backs. The AVF reduction results achieved by using the proposed techniques are illustrated in the Fig 8. As the figure reveal, on the average across all benchmarks AVF is reduced by more than 32% while individual benchmarks, such as the hmmer, show benefits of as high as 42%.

As a conlusion in this paper, we analyzed the simulation results for the vulnerability of the RAT against soft errors and investigated the vulnerability of the RAT checkpoints for RAM array structure accessed as in RAB. Cycle accurate simulations revealed that average RAT vulnerability is 25% and average RAT checkpoint vulnerability is 5.66% for aforementioned SPEC 2006 benchmarks. Then, we proposed two techniques for error detection and correction utilizing the redundant information in sequential checkpoints. On the average on-the-fly check is capable of detecting 41.5% of the errors by checking the current checkpoint with the previous. On-demand check is capable of detecting almost half of the errors on a roll-back and of correcting one third of the errors occurred in one of the checkpoint in neighborhood of the rolled-back checkpoint. We have full custom designed the RAT table and checkpoint table and comparators for proposed techniques. Adding an extra control bit for error check brings little area and delay overhead and small energy consumption.

VII. ACKNOWLEDGEMENTS

This work was supported in part by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant 112E004 and in framework of COST ICT Action 1103.

REFERENCES

[1] E. Safi, A. Moshovos, and A. Veneris, “A physical level study and optimization of CAM-based checkpointed register alias table,” in *Proc. IEEE Int. Symp. Low Power Electron. Des.*, pp. 233–236, Aug. 2008.

[2] E. Safi, P. Akl, A. Moshovos, A. Veneris, and A. Arapoyianni, “On the Latency, Energy and Area of Checkpointed, Superscalar Register Alias Tables,” in *Proc. IEEE Int. Symp. Low Power Electron. Des*, 2007

[3] E. Safi et al., “On the Latency and Energy of Checkpointed Superscalar Register Alias Tables,” *IEEE Transactions on VLSI* vol.18, no.3, 2010.

[4] H.Zeng, M. T. Yourst, and K. Ghose, “An energy-efficient checkpointing mechanism for out of order commit processor”, in *Proc. 14th ACM/IEEE Intl. Sym. on Low Power Electron. Des.*, 2009.

[5] K. C. Yeager, “The MIPS R10000 Superscalar Microprocessor”, in *Proc. of the 29th Int. Symp. on Microarchitecture* vol. 16, no. 2, 1996.

[6] N. Binkert et.al., “The gem5 simulator,” *ACM SIGARCH Computer Architecture News*, vol.39, no.2. pp.1–7, May 2011.

[7] P. Montesinos, W. Liu, J. Torrellas, “Using Register Lifetime Predictions to Protect Register Files Against Soft Errors,” 37th International Conference on Dependable Systems and Networks 2007

[8] R. C. Baumann, “Soft errors in advanced computer systems,” IEEE Des. Test. Comput., vol. 22, no. 3, pp. 258–266, May/Jun. 2005.

[9] R.E. Kessler, “The Alpha 21264 Microprocessor,” *Proceedings of the 32nd International Symposium on Microarchitecture*, vol. 19, 1999

[10] R.E. Lyons and W. Vanderkulk, “The use of triple-modular redundancy to improve computer reliability,” *IBM Journal of R&D*, vol.6, no2, 1962

[11] S. S. Mukherjee et al. “A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor”, in *Proc of the 36 Int. Symp. on Microarchitecture,03*

[12] D. J. Sorin et al. “ARGUS: Low-Cost, Comprehensive Error Detection in Simple Cores”, *IEEE Micro 2008*

[13] T. N. Buti et. al., “Organization and implementation of the register-renaming mapper for out-of-order IBM POWER4 processors”, *IBM Journal of Research and Development*, vol.49, no.1, pp. 167-188, 2005.

[14] W. Hwu and Y. N. Patt. Checkpoint repair for out-of-order execution machines. In *Proceedings of the 14th ISCA, June 1987*

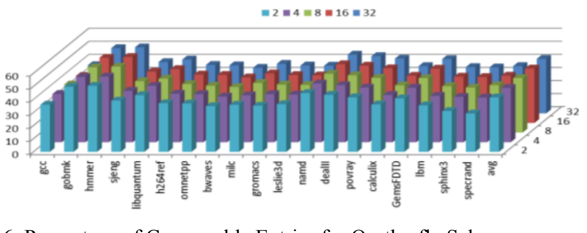


Fig. 6 Percentage of Comparable Entries for On-the-fly Scheme

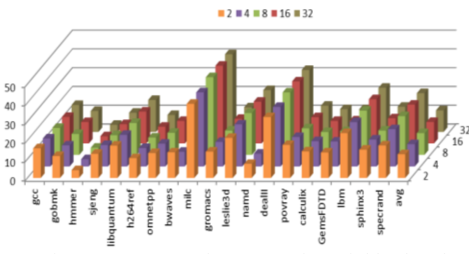
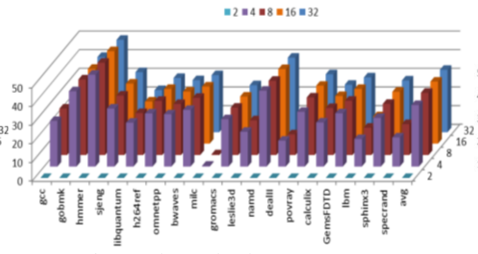



Fig. 7. One CE and Two CEs in Neighborhood Percentage for On-demand Scheme

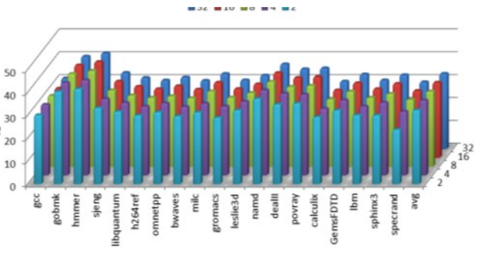


Fig. 8. Total AVF Reduction by Proposed Techniques