Reliability-Aware Exceptions: Tolerating Intermittent Faults in Microprocessor Array Structures

Waleed Dweik, Murali Annavaram, and Michel Dubois
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, United States of America

***Abstract*—In future technology nodes, reliability is expected to become a first-order design constraint. Faults encountered in a chip can be classified into three categories: transient, intermittent, and permanent. Fault classification allows a chip to take the appropriate corrective action. Mechanisms have been proposed to distinguish transient from non-transient faults where all non-transient faults are handled as permanent. Intermittent faults induced by wearout phenomena have become the dominant reliability concern in nanoscale technology, yet there is no mechanism that provides finer classification of non-transient faults into intermittent and permanent faults. In this paper, we present a new class of exceptions called Reliability-Aware Exceptions (RAEs) which provide the ability to distinguish intermittent faults in microprocessor array structures. The RAE handlers have the ability to manipulate microprocessor array structures to recover from all three categories of faults. Using RAEs, we demonstrate that the reliability of two representative microarchitecture structures, load/store queue and reorder buffer in an out-of-order processor, is improved by average factors of 1.3 and 1.95, respectively.**

***Keywords*-intermittent fault; array strucutre; fault injection; de-configuration**

I. INTRODUCTION

Technology scaling has led to variations in device characteristics resulting in a range of susceptibilities. Faults encountered during the in-field operation can be classified into three categories: transient, intermittent and permanent. Transient faults occur once and then disappear. Permanent faults are persistent. Intermittent faults oscillate between periods of erroneous activity and dormancy, depending on various operating conditions such as temperature and voltage.

Intermittent faults are becoming dominant factors that limit chip lifetime in nanoscale technology nodes [7][15]. Existing detection and recovery approaches classify faults as transients or permanents [2][13]. This crude classification results in unnecessary performance degradation and increases the probability of permanent failure due to excessive usage.

Modern microprocessors contain large array structures in the form of buffers and tables, such as instruction fetch queue (IFQ), load/store queue (LSQ), reorder buffer (ROB), reservation stations (RS), and branch history/prediction tables. Unlike computational structures, array structures are highly stressed because they get read or written continuously. Even during idle periods, array structures are still considered under stress because they continue to store the same bit values which have negative-bias temperature instability (NBTI) impact on the PMOS devices. This makes the microprocessor array structures highly susceptible to intermittent faults.

978-3-9815370-2-4/DATE14/©2014 EDAA

In this paper we propose Reliability-Aware Exceptions (RAEs), a special class of exceptions that enable software-directed handling of faults detected in the microprocessor array structures. RAEs are used in conjunction with hardware fault detection mechanisms to improve microprocessor lifetime. The novelty of RAEs is the ability to distinguish intermittent faults from transient and permanent faults and to provide appropriate corrective actions for the three fault categories. For transient faults, no corrective action is required and it is sufficient to roll-back the execution to the faulting instruction. For intermittent (permanent) faults, before the execution is rolled-back, RAEs exploit the inherent redundancy in the microprocessor array structures to temporarily (permanently) de-configure the faulty entries.

II. WHY TO DISTINGUISH INTERMITTENT FAULTS?

A fault analysis study of the memory subsystems of 257 servers revealed that 6.2% of the subsystems experienced intermittent faults [7]. A more recent study analyzed the fault logs reported by the Microsoft Windows Error Reporting program from one million consumer PCs [15]. 39% of the CPU faults turned out to be intermittent. Furthermore, it has been found that Soft Breakdown (SBD) in the device dielectric causes intermittent erratic fluctuations in the minimum voltage of 90nm SRAM cells [1]. Finally, interconnect resistance increases due to electro-migration (EM) which causes intermittent timing violations under high temperatures [8]. Based on the above studies, intermittent faults are becoming a major reliability concern in modern microprocessors.

Intermittent faults have two important features that distinguish them from transient and permanent faults. First, an intermittent fault recurrently occurs in the same location which is different from a random bit flip as in the case of a transient fault. Second, intermittent faults oscillate between periods of activation and deactivation (i.e. bursty behavior) according to the current operating conditions, whereas transient faults are one-time events and permanent faults are continuously activated. This indicates that handling intermittent faults as

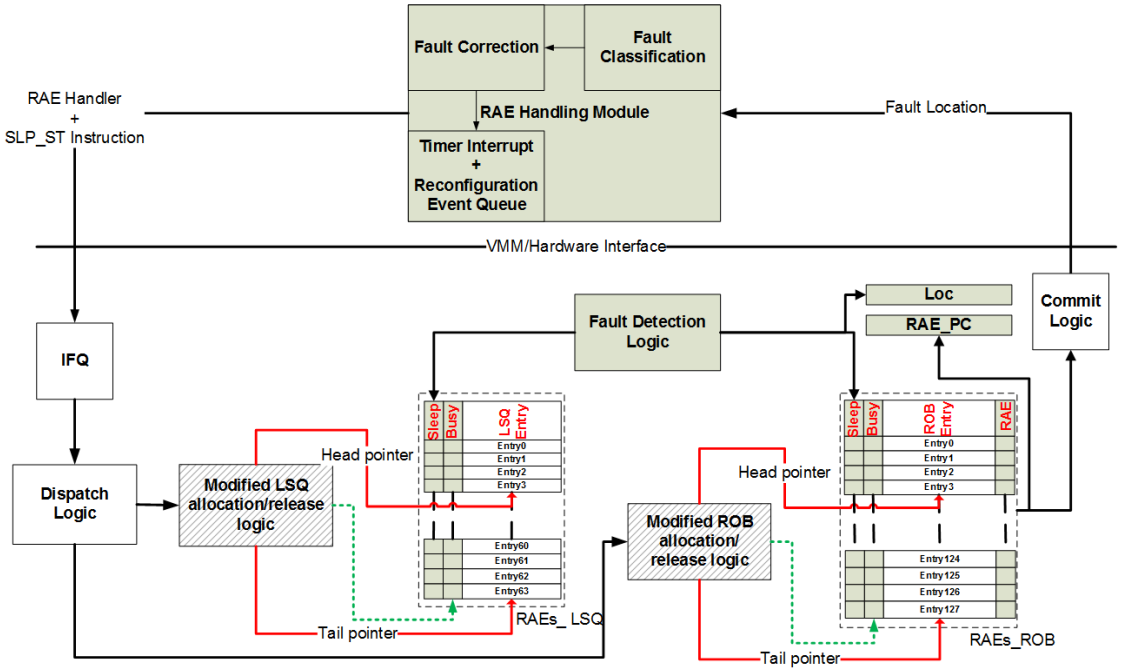

Fig. 1. RAEs Implementation

transients (i.e. flush and re-execute) will cause excessive flushing and degrade the performance significantly. On the other hand, handling intermittent faults as permanents will cause pre-mature permanent resource de-configuration. Hence, the effectiveness of fault handling solutions can be significantly improved if they can distinguish and handle intermittent faults as a separate fault category.

A major root cause of intermittent faults is device wearout. The most important wearout phenomena include electro-migration (EM), Negative Bias Temperature Instability (NBTI), and Time Dependent Dielectric Breakdown (TDDB). These phenomena manifest as timing violations activated under abnormal temperature, stress and voltage conditions. In the case of NBTI, the device deterioration can be reversed and the intermittent fault can be eliminated by removing the stress conditions [11]. In the case of EM and TDDB, removing the stress conditions only deactivates the intermittent faults but does not reverse the degradation. These aging manifestations occur slowly over time; hence, it is necessary to develop low-cost solutions to handle wearout-induced intermittent faults. This is exactly what RAEs do.

III. Related Work

A threshold-based mechanism to discriminate transient from non-transient faults is described in [2]. Bondavalli et al. introduced a count-and-threshold analytical model called α-count. At regular time intervals, hardware detection mechanisms generate a single bit data for each system component. These bits are used to produce α-value for each component which in turn is compared with a set of pre-defined thresholds to classify each component as faulty or healthy.

Bower et al. proposed Self-Repairing Array Structures (SRAS) [3]. SRAS map-out entries that experience permanent faults. SRAS use a handful of check rows such that every write operation to a row is duplicated to a check row. Then data is read from both rows and compared to detect faults.

Wells, Chakraborty, and Sohi proposed pausing core execution to adapt with intermittent faults in multicore systems [21]. Suspending the use of the core helps deactivate intermittent faults in all parts of the core.

RAEs differ from previous work in three folds: first, RAEs use a historical fault log and faults inter-arrival time to classify a fault as transient, intermittent, or permanent. Second, RAEs handle intermittent faults using temporal de-configuration which helps *deactivate* the intermittent faults. Moreover, temporal de-configuration helps *remove* NBTI-induced intermittent faults by recovering the device state. Third, RAEs are implemented at the granularity of array structure entries; hence, do not require the entire core operation to be suspended.

IV. Building Blocks for RAEs Design

A. Fault Detection

Fig. 1 gives an overview of how RAEs can protect two representative array structures of an Out-of-Order processor, namely the load/store queue (LSQ) and the reorder buffer (ROB). The hardware/software additions necessary for RAEs are highlighted in solid color and the modified blocks are shaded with stripes.

The first step in RAEs is to detect faults as soon as they occur. Transient, intermittent, and permanent faults manifest either as bit flips or as timing violations. Commercial fault-tolerant microprocessors (e.g. IBM POWER6 and z10 microprocessors) use parity to protect the internal array structures against bit flips (excluding caches and register files) [16][19]. Similarly, RAEs leverage the existing parity to detect

bit flips assuming a single fault model. In addition, RAEs rely on RazorII to detect timing violations [10].

Whenever an array structure entry is read by an instruction, the detection mechanisms are activated to validate that the data read is fault free. Fault detection is done off the critical path. Whenever a fault is detected, three actions are required: first, the faulting instruction is turned into a nop. Second, the detection hardware sets a special bit in the ROB entry of the faulting instruction. The special bit is called the "*RAE*" bit and it indicates that there is a reliability-aware exception associated with this instruction. Third, the identification number of the faulty array structure entry (ASE_ID) and the ROB entry number (ROB_ID) of the faulting instruction are stored in a special register called "*loc*". ASE_ID is used later to classify the fault and to de-configure the faulty entry when needed. ROB_ID is used to check if the faulting instruction is at the top of the ROB and ready to commit.

Since instructions may trigger exceptions out-of-order, it is necessary to handle the exceptions in process order by allowing only older instructions to overwrite the "*loc*". Fig. 1 shows each ROB entry augmented with an "*RAE*" bit and it shows the "*loc*" register. To reduce clutter, fault detection mechanisms are not shown in Fig. 1. The "*RAE*" bits and the "*loc*" register are assumed to be built fault-free using device sizing or error correction mechanisms.

B. Disabling the Faulty Entry

Once a fault is detected, the faulty array structure entry is disabled to ensure that no following instructions are allocated to the same entry until the exception handler has dealt with the fault. *Disabling* a specific entry means "to prevent that entry from further use", but the entry is NOT turned OFF.

Microprocessor array structures can be categorized into: circular buffer structures (e.g. IFQ, LSQ, and ROB) and tabular array structures (e.g. RS and branch history and prediction tables). To disable faulty entries of circular structures, we augment every entry with a "*busy*" bit. The faulty entry is disabled by setting its "*busy*" bit to one [3]. To disable faulty entries of tabular arrays, we mark them in the allocation/release logic as in-use [4].

Fig. 1 shows LSQ and ROB entries augmented with "*busy*" bits which are assumed to be fault-free. In addition, the allocation/release logic of LSQ and ROB need to be modified to consult the "*busy*" bit array before allocating new entries.

C. RAE Handler: Fault Classification

When the instruction with an RAE reaches the top of the ROB and is ready to commit, the CPU is flushed and the RAE handler is invoked in order to classify the fault and choose the most appropriate corrective action.

RAEs classification relies on mining a historical fault log. Every array structure entry has a single record in the log which contains: a fault counter to keep track of the number of faults detected in the entry and a timestamp of the latest fault detected in the same entry. Once a reliability exception is raised, the RAE handler reads the "*loc*" register to obtain the ASE_ID and uses it to access the corresponding record in the fault log. To classify the current fault, the RAE handler compares the current timestamp with the record's timestamp. There are two possible scenarios: first, the record's timestamp is zero (i.e. the current fault is the very first fault detected in the entry) *or* the difference between the two timestamps is greater than or equal to the inter-arrival time between transient faults. In this scenario, the current fault is optimistically classified as transient.

Second, the record's timestamp is greater than zero (i.e. one or more faults were previously detected in the entry) *and* the difference between the two timestamps is less than the inter-arrival time between transient faults. In this scenario, the fault is either classified as intermittent or permanent based on the record's fault counter value. If the counter value is less than a pre-determined *Intermittent-Fault-Threshold*, the fault is classified as intermittent; otherwise, the fault is classified as permanent. To avoid misclassifying intermittent faults as permanents, the fault counters of all fault log records are periodically reset to zero.

The inter-arrival time between transient faults, the *Intermittent-Fault-Threshold*, and the period to clear the log's fault counters are design parameters that depend on the current process technology node. For evaluation purposes, the values of the three parameters are discussed in section V below.

Accessing the software fault log for fault classification is a slow process. But under the assumption that faults are rare events compared to the normal operational window of a processor, we can afford to handle fault scenarios using slow software handlers in order to save area and power consumed by hardware-based mechanisms. Moreover, the size of each log record is 12 bytes which makes the total log size of LSQ and ROB 2KBytes. Hence, the fault log can easily fit in the main memory or even in the last level cache to speed up fault classification.

D. RAE Handler: Fault Correction

After fault classification is done, one of the three following corrective actions is undertaken: (1) Transient_Action: enable the faulty entry. (2) Intermittent_Action: temporarily de-configure the faulty entry. De-configuring an entry means "to turn OFF the entry by power gating". Temporal de-configuration helps deactivate the intermittent fault. More importantly, in the case of NBTI-induced faults, temporal de-configuration helps PMOS transistors recover to their original state (i.e. eliminates the intermittent fault) [11]. (3) Permanent_Action: permanently de-configure the faulty entry.

Notice that when the entries surrounding the faulty entry experience high activity, the effect of temporal de-configuration may be limited (i.e. the fault may persist). Even in such cases, de-configuration is still useful because it avoids using the faulty entry and consequently reduces the performance overhead caused by excessive flushing.

After fault correction is complete, the execution is resumed starting from the faulting instruction. In a processor that supports speculative execution, like the one we used for RAEs evaluation, every instruction carries its program counter (PC) in its designated ROB entry. This PC is used for re-execution

in the case of a branch misprediction or a software exception. RAEs' handlers cannot trust the PC of the faulting instruction obtained from the ROB because the ROB contents might be faulty. As a result, we augment the commit stage with a fault-free register called "*RAE_PC*" as shown in Fig. 1. "*RAE_PC*" holds the PC value of the instruction following the last correctly committed instruction. In case the last committed instruction was a control instruction, the "*RAE_PC*" holds the PC of the target instruction. Otherwise, the "*RAE_PC*" holds the PC of the next instruction in program order.

E. De-configuration Approach

The temporal or permanent de-configuration of the faulty entry in the case of intermittent or permanent faults is achieved through sleep transistors [17]. De-configuration is done at the granularity of a single entry. Hence, each entry has its own sleep transistor and all entries are augmented with "*sleep*" bits to drive their sleep transistors. "*Sleep*" bits are assumed to be fault-free.

We introduce a new instruction called SLP_SET. For intermittent and permanent faults, SLP_SET is used by the RAE handler to de-configure the faulty entry by setting its "*sleep*" bit to one. For intermittent faults only, a reconfiguration event must be scheduled when the fault is expected to be no longer active. RAEs keep a sorted queue of the reconfiguration events for all entries that are currently de-configured due to intermittent faults, such that the reconfiguration event that will occur earliest is at the head of the queue. When a reconfiguration event is due, the RAE handler also uses SLP_SET to reconfigure the corresponding entry by resetting its "*busy*" and "*sleep*" bits.

As mentioned in subsection IV.B above, once a fault is detected, the faulty entry is disabled from further use by setting its "*busy*" bit. In case of transient faults, the faulty entry needs to be re-enabled by resetting its "*busy*" bit. The RAE handler leverages SLP_SET instruction to achieve that as well. SLP_SET takes fault location and fault type as source operands. For de-configuration purposes, the fault type operand of the SLP_SET is set to intermittent or permanent which essentially sets the "*sleep*" bit to one. For reconfiguration and enabling purposes, the fault type operand of the SLP_SET is set to transient, which essentially resets both "*busy*" and "*sleep*" bits to zero.

V. Evaluation Methodology

To evaluate how much reliability improvement RAEs can provide for the microprocessor array structures, we have developed a simulation infrastructure that models various types of faults and relies on accelerated fault injections.

The simulation infrastructure is based on the Simplescalar tool [6] integrated with Wattch [5] and HotSpot [12]. The Simplescalar is augmented to keep track of the activity factors of different array structures (i.e. number of read/write operations). The activity factors are plugged into the power models of Wattch to estimate the power consumptions of the array structures. The power trace is then fed to Hotspot to compute the temperature of each structure based on a given floor plan.

The processor model simulated is a 4-way Out-of-Order processor with 64-entry LSQ and 128-entry ROB. The processor model stores the speculative results from instruction execution in the ROB. In our evaluation, we compare three different processors: *base* (i.e. No RAEs support), *large* (i.e. No RAEs support, but LSQ and ROB are built with bigger transistors which have lower failure probability), and *RAEs* (i.e. with RAEs support). *Base* and *large* processors cannot distinguish intermittent faults and they classify a fault as either transient or permanent. For transient faults, the execution is simply rolled-back, whereas permanent faults are handled via permanent de-configuration.

For *RAEs* simulations, the *Intermittent-Fault-Threshold* and the period to clear the logs' fault counters are chosen based on the comprehensive experimental study in [7]. The results of the study indicate that an intermittent fault in a memory structure entry is activated up to 15 times every day. So, we set the *Intermittent-Fault-Threshold* to 15 and the logs' fault counters are reset to zero every day.

The transistors of the LSQ and ROB structures in the *large* processor are 7% bigger than the transistors used in *base* and *RAEs* processors. We have chosen 7% because it is the estimated area overhead for supporting RAEs, as will be shown in subsection VI.C below. According to [22], a transistor failure probability is linearly proportional to the size of its gate. Hence, the failure probability of LSQ and ROB entries in the *large* processor is expected to be 7% lower than *base* and *RAEs* processors. We use an accelerated fault injection methodology to do the evaluation. It has three main parameters: fault rate, fault type, and fault model.

A. Fault Rate

The total fault rate of an array structure is the summation of the transient, intermittent, and permanent fault rates of the structure. In 45nm process technology, the transient fault rate per bit is 10^{-3} FIT [20]. In our evaluation infrastructure, the LSQ has 64 72-bit wide entries and the ROB has 128 72-bit wide entries. Hence, LSQ's transient fault rate is 5 FIT and ROB's transient fault rate is 10 FIT. The transient fault rate is used to compute the average inter-arrival time between transient faults needed by RAEs fault classification.

For intermittent fault rate, we rely on the reliability modeling framework presented in [18]. The framework uses a new concept called the failure-in-time of reference circuit (FORC) to compute the FIT rates of LSQ and ROB due to EM, NBTI, and TDDB. These FORC models depend on structures' sizes, environmental conditions (e.g. temperature), and stress conditions measured by the probabilities of having logic 0 or logic 1 in a cell. Our simulation infrastructure measures the average temperatures and stress conditions for LSQ and ROB while running SPEC CPU2006 benchmarks. Then the measured values are used to compute intermittent fault rate of LSQ and ROB as 275 and 550 FIT, respectively.

Current microprocessors are expected to have a mean-time-to-failure (MTTF) of 5 years; this translates to a cumulative intermittent and permanent fault rates of 22831 FIT. Assuming this cumulative fault rate is uniformly distributed across the microprocessor footprint, we deduce the

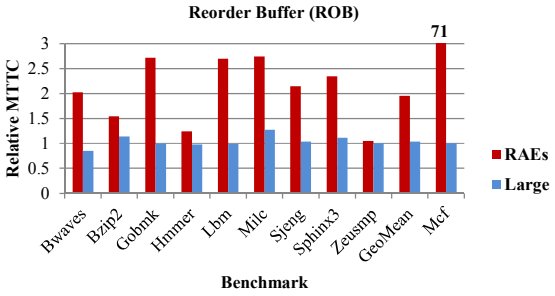

Fig. 2. Relative ROB MTTC

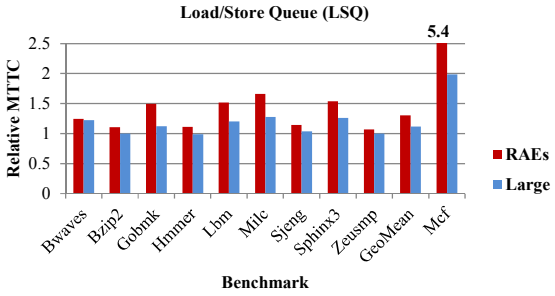

Fig. 3. Relative LSQ MTTC

permanent fault rates of LSQ and ROB as 38 and 76 FIT, respectively. So, the total fault rates of LSQ and ROB are 318 and 636 failures in billion hours, respectively. Assuming a clock frequency of 1GHz, the failure probabilities per cycle for LSQ and ROB are ($8.83*10^{-20}$) and ($17.7*10^{-20}$).

The failure probabilities of LSQ and ROB indicate that faults are rare events compared to the processor normal operational window. Hence, we need to accelerate the occurrences of faults in our simulations. An acceleration factor of 10^{13} is chosen because it allows a couple of hundred faults to be injected during the simulation of each benchmark. Therefore, the simulated failure probabilities per cycle for LSQ and ROB in *base* and *RAEs* processors are ($8.83*10^{-7}$) and ($17.7*10^{-7}$), respectively. In the *large* processor, the simulated failure probabilities per cycle for LSQ and ROB are ($8.2*10^{-7}$) and ($16.5*10^{-7}$). Similarly, the average inter-arrival time between transient faults and the period to clear the logs' fault counters are scaled according to the acceleration factor.

B. Fault Type

Whenever a fault is to be injected, the fault type needs to be specified. For each array structure, the probability of transient, intermittent, and permanent faults can be computed using the failure rate value of each type given in subsection V.A above. For example the probability of transient faults in the LSQ ($P_{LSQ\text{-}Transient}$) can be computed as follows:

$$P_{LSQ-Transient} = FIT_{LSQ-Transient}/FIT_{LSQ-Total} \qquad (1)$$

Where:

$$FIT_{LSQ-Total} = FIT_{LSQ-Transient} + FIT_{LSQ-Intermittent} + FIT_{LSQ-Permanent}$$

A random variable with standard uniform distribution *U(0,1)* is generated every time a fault is to be injected. The random variable is used to determine the fault type according to the probability of each fault category computed as in (1).

C. Fault Model and Period

The last aspect of the fault injection setup is to decide how to model each fault type and how long it will persist. Transient faults are modeled as bit flips and they persist for a single cycle. Permanent faults are modeled as stuck-at and they persist for the entire simulation. Intermittent faults are modeled as timing violations which manifest as bit errors when the faulty entry is read. Due to the lack of information about the activation and deactivation periods of intermittent faults, we rely on our empirical results which indicate an average continuous stress time in the order of thousands of cycles. Hence, we randomly pick among three intermittent fault periods of 1000, 2000, and 4000 cycles.

For intermittent faults we also need to determine the temporary de-configuration period. Assuming a clock frequency of 1GHz and based on the fact that intermittent faults deactivate in the first second after stress is removed [14], the faulty entry is de-configured for 1 billion cycles. Once the de-configuration period expires, the entry is reconfigured into normal operation as we described in subsection IV.E above.

VI. Experimental Results

For our simulation experiments, we have chosen ten SPEC CPU2006 benchmarks. We fast-forwarded through the first 300 million instructions then ran detailed simulations for three billion instructions.

A. Reliability Evaluation

In order to compare the reliability of the LSQ and ROB in the three processors, we define a new reliability metric called mean time to crash (MTTC). MTTC is the average time for all entries in a structure to become faulty so as to cause the system to crash.

To compute the MTTC of a structure, we denote the number of cycles until all its entries are faulty as *N*. Assuming that every clock cycle, one entry may become faulty with a probability *f*, then *N* is essentially a random variable with a negative binomial distribution NB(r,q). Where *r* is the number of the remaining non-faulty entries in the structure and *q* is the probability of not having a fault in the entry that is being accessed in the current cycle (i.e. *q = 1-f*). The expected value of *N* is the MTTC of the structure and is computed as follows:

$$E(N) = MTTC = (q \times r)/(1-q) \qquad (2)$$

Fig. 2 shows the MTTC of the ROB in the *RAEs* and *large* processors relative to the *base* processor at the end of the simulations. The geometric mean for all benchmarks, except Mcf, shows that the *RAEs* improve the MTTC of the ROB with respect to the *base* and *large* processors by factors of 1.95 and 1.82. This improvement is mainly due to the ability of the *RAEs* processor to identify intermittent faults and recover from them using temporal de-configuration. Mcf is

excluded from the mean calculation because *base* and *large* processors exhausted their ROB structures before completing all three billion instructions. For Mcf, the *RAEs* processor achieved a MTTC improvement factor of 71.

Another observation in Fig. 2 is that for some benchmarks, the *large* processor performs the same or even worse than the *base* processor despite the lower fault injection rate. This is due to the randomization effect in determining the fault type as described in subsection V.B above. As a result, more intermittent or permanent faults could get injected in the *large* processor which makes it look worse than the *base* processor. The randomization effect is expected to disappear as the benchmark run time increases.

Fig. 3 shows the MTTC of the LSQ structure in the *RAEs* and *large* processors relative to the *base* processor at the end of the simulations. The geometric mean for all benchmarks, except Mcf, shows that the *RAEs* improve the MTTC of the LSQ with respect to *base* and *large* processors by factors of 1.30 and 1.18. LSQ utilization is typically less than ROB because it is only accessed by memory-access instructions. Hence, smaller improvement factors are achieved for LSQ.

B. Performance Evaluation

The performance of the three processors is measured by their execution time. On average, the *RAEs* processor achieves a speed up of 5% and 4% with respect to *base* and *large* processors. This is because the *RAEs* processor does not permanently de-configure LSQ and ROB entries with intermittent faults.

C. Area Overhead

In order to protect the LSQ and the ROB with RAEs, every entry needs to be augmented with sleep, busy, and parity bits. In addition, ROB entries are augmented with RAE bits as shown in Fig. 1. This adds up to a total of 704 additional bits, which require 4224 transistors. The parity checkers require 840 transistors. The "*loc*" and "*RAE_PC*" registers are 18 and 32-bit wide, respectively. Hence, they require 300 transistors. Thus, the total number of transistors required to protect LSQ and ROB using RAEs is 5364 which is 6.5% of the total number of transistors used to build the base LSQ and ROB.

VII. Conclusions

In this paper we introduce Reliability-Aware Exceptions (RAEs), a new class of exceptions that use software handlers to deal with hardware faults in microprocessor array structures. RAEs have the ability to identify intermittent faults and handle them using temporal de-configuration.

Our experimental results show that RAEs significantly improve the reliability of LSQ and ROB. Although our experiments focused on two major array structures, we expect to get the same advantages when applying RAEs to other circular buffer structures and tabular array structures.

Acknowledgments

This work was supported by DARPA-PERFECT-HR0011-12-2-0020 and NSF grants NSF-1219186, NSF-CAREER-0954211, NSF-0834798.

References

[1] Agostinelli, M. et al, "Erratic fluctuations of sram cache vmin at the 90nm process technology node," *IEDM Technical Digest*, Dec. 2005.

[2] Bondavalli, A.; Chiaradonna, S.; di Giandomenico, F.; Grandoni, F., "Threshold-based mechanisms to discriminate transient from intermittent faults," *IEEE Transactions on Computers*, Mar. 2000.

[3] Bower, F.A.; Shealy, P.G.; Ozev, S.; Sorin, D.J., "Tolerating hard faults in microprocessor array structures," *International Conference on Dependable Systems and Networks*, July 2004.

[4] Bower, F.A.; Sorin, D.J.; Ozev, S., "A mechanism for online diagnosis of hard faults in microprocessors," *38th Annual IEEE/ACM International Symposium on Microarchitecture*, Nov. 2005.

[5] Brooks, D.; Tiwari, V.; Martonosi, M., "Wattch: a framework for architectural-level power analysis and optimizations," *27th International Symposium on Computer Architecture*, June 2000.

[6] Burger, D.; T. M. Austin, T.M., "The SimpleScalar Tool Set, Version 2.0," *Computer Architecture News*, June 1997.

[7] Constantinescu, C., "Intermittent faults and effects on reliability of integrated circuits," *Annual Reliability and Maintainability Symposium*, Jan. 2008.

[8] Constantinescu, C., "Intermittent Faults in VLSI Circuits", *IEEE Workshop on System Effects of Logic Soft Errors*, Apr. 2006.

[9] Cristal, A.; Santana, O.J.; Valero, M.; Martínez, J.F., "Toward kilo-instruction processors," *ACM Transactions on Architecture and Code Optimization (TACO)*, Dec. 2004.

[10] Das, S. et al, "RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance," *IEEE Journal of Solid-State Circuits*, Jan. 2009.

[11] Ershov, M. et al, "Dynamic recovery of negative bias temperature instability in p-type metal–oxide–semiconductor field-effect transistors," *Applied Physics Letters*, Aug. 2003.

[12] Wei Huang; Sankaranarayanan, K.; Skadron, K.; Ribando, R.J.; Stan, M.R., "Accurate, Pre-RTL Temperature-Aware Design Using a Parameterized, Geometric Thermal Model," *IEEE Transactions on Computers*, Sept. 2008.

[13] Li, M.; Ramachandran, P.; Sahoo, S.K.; Adve, S.V.; Zhou, Y., "Understanding the propagation of hard errors to software and implications for resilient system design," *Computer Architecture News*, Mar. 2008.

[14] Mahapatra, S., "Negative Bias Temperature Instability (NBTI) in p-MOSFETs: Characterization, Material/Process Dependence and Predictive Modeling," *http://nanohub.org/resources/13613*.

[15] Nightingale, E.B.; Douceur, J.R.; Orgovan, V., "Cycles, cells and platters: an empirical analysis of hardware failures on a million consumer PCs," *Sixth Conference on Computer systems*, Apr. 2011.

[16] Reick, K. et al, "Fault-Tolerant Design of the IBM Power6 Microprocessor," *IEEE Micro*, March-April 2008.

[17] Shi, K.; Howard, D., "Sleep Transistor Design and Implementation - Simple Concepts Yet Challenges To Be Optimum," *International Symposium on VLSI Design, Automation and Test*, Apr. 2006.

[18] Shin, J.; Zyuban, V.; Zhigang Hu; Rivers, J.A.; Bose, P., "A Framework for Architecture-Level Lifetime Reliability Modeling," *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2007.

[19] Shum, C. -L K et al, "Design and microarchitecture of the IBM System z10 microprocessor," *IBM Journal of Research and Development*, Jan. 2009.

[20] Slayman, C., "Soft error trends and mitigation techniques in memory devices," *Annual Reliability and Maintainability Symposium*, Jan. 2011.

[21] Wells, P.M.; Chakraborty, K.; Sohi, G.S., "Adapting to Intermittent Faults in Future Multicore Systems," *16th International Conference on Parallel Architecture and Compilation Techniques*, Sept. 2007.

[22] Quming Zhou; Mohanram, K., "Gate sizing to radiation harden combinational logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Jan. 2006.