Proactive NBTI Mitigation for Busy Functional Units in Out-of-Order Microprocessors

Lin Li*, Youtao Zhang†, Jun Yang*, Jianhua Zhao‡
*Dept of ECE, University of Pittsburgh; †Dept of CS, University of Pittsburgh; ‡Dept of CS, Nanjing University
*lil53@pitt.edu, †zhangyt@cs.pitt.edu, *juy9@pitt.edu, ‡zhaojh@nju.edu.cn

***Abstract*—Due to fast technology scaling, negative bias temperature instability (NBTI) has become a major reliability concern in designing modern integrated circuits. In this paper, we present a simple and proactive NBTI recovery scheme targeting at critical and busy functional units with storage cells in modern microprocessors. Existing schemes have limitations when recovering these functional units. By exploiting the idle time of busy functional units at per-buffer-entry level, our scheme achieves on average 5.57× MTTF (Mean Time To Failure) improvement at the cost of <1% IPC degradation and <1% area overhead.**

I. Introduction

With technology scaling to the realm of nanoscale, the reliability of modern integrated circuits is jeopardized by various failure sources such as negative bias temperature instability (NBTI), electromigration, gate oxide breakdown, stress migration, time dependent dielectric breakdown (TDDB) and thermal cycling [13]. NBTI is of particular concern as it is the dominant failure source for PMOS transistors — when being stressed by a negative gate voltage, a PMOS transistor exhibits an increase of its threshold voltage. NBTI gradually slows down the gate transition speed, and eventually causes the transistor failing to meet the timing or stability constraint in the corresponding integrated circuit. That is, NBTI degrades circuit reliability.

Recently several recovery schemes have been proposed to mitigate the NBTI-induced reliability loss. At the circuit level, [14] adopted oversized PMOS transistors to provide timing guard-band to offset the gradual speed loss. [11] reduced the supply voltage to mitigate the voltage stress. Since a PMOS transistor is under stress only when the input is "0", and weakly recovers from NBTI when the input is "1", these circuit level methods failed to handle the dynamic NBTI degradation induced by the varying "0" distribution of PMOS inputs. At the architecture level, [2] proposed to maximize the probability of a PMOS transistor having input "1" by setting special inputs to ALU and other combinatorial circuits during their idle cycles. For storage cells, due to their symmetric structures, the stored values may be inverted during idle periods to achieve balanced probability of 0s and 1s [5] [2]. A stronger and more proactive approach was later proposed for memory cells in L2 caches [12]. There, the virtual power on one side of a cache bank is shut down periodically, and a positive gate-to-source voltage (V_{gs}) is applied to the cache bank to proactively recover their PMOS transistors. This technique requires spare elements (i.e. an extra cache bank) for data backup because the elements under recovery lose their states.

In this paper, we perform strong NBTI recovery on the PMOS transistors of core functional units (FUs) such as reservation stations (RS), reorder buffers (ROB) and physical registers (PR). Our contributions are summarized as follows:

- We successfully identify the recovery opportunities for these critical and busy FUs in modern microprocessors. We observed that the entries in those FUs present on average 43∼76% idle cycles for SPEC2000 benchmark programs. It is preferrable to perform proactive NBTI-recovery during idle cycles as it does not need spare FUs to back up the old contents, and achieves stronger recovery effect than static schemes [2].
- We design a per-buffer-entry based recovery logic to effectively exploit the identified recovery opportunities. For the 43∼76% cycles identified as idle in aforementioned FUs, our scheme utilizes 27∼61% cycles for proactive recovery. We have modeled our design using HSPICE and CACTI tools, and observed <1% performance degradation and <1% area overhead.
- The design greatly improves the reliability of the aforementioned FUs, and extends their lifetime significantly. The experimental results showed that the proposed design achieves at least 4.44× and on average 5.57× MTTF improvement.

The remainder of this paper is organized as follows. Section II discusses the background of NBTI and the proactive NBTI recovery for L2 caches. Section III studies the proactive NBTI recovery opportunities for critical and busy FUs within processors. Section IV presents the design details and estimates the latency and area overheads. Section V discusses our evaluation methodology and analyzes the experimental results. Finally, Section VI concludes the paper.

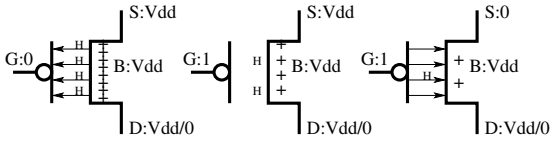
II. Background

A. NBTI Basics

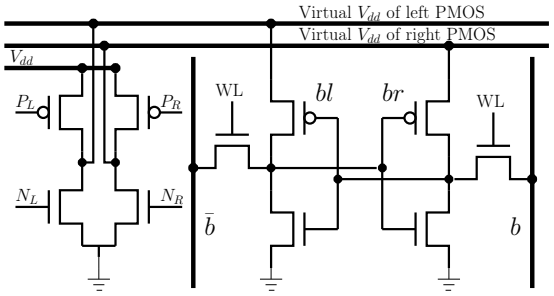
NBTI occurs in PMOS transistors when the input of the gate is "0" at elevated temperature, as shown in the leftmost case in Fig. 1(a). When the gate is under negative gate-source bias i.e., $V_{gs} = -V_{dd}$, a series of physical-chemical processes are triggered such that positively charged traps are created near the gate oxide [16]. These positive traps raise the threshold voltage, which gradually slows down the gate transition speed and reduces the noise margin of the circuit. Eventually the circuit will fail as the PMOS transistor becomes too slow to meet the timing or stability requirement. Studies showed that NBTI-induced failure becomes more severe under higher temperature and stronger electric field [13].

Different recovery modes have been investigated to increase the lifetime of PMOS transistors. The first recovery mode, referred as *moderate recovery*, is shown in the middle case of Fig. 1(a). It happens when the input of the gate is "1" with normal V_{dd}. This is the mild recovery used in [2]. When the input changes to "1", the negative bias stress is removed, the traps in the gate oxide are healed, and the NBTI degradation on the V_{th} of PMOS can be recovered partially. Note that when the input switches between logic "0" and "1" on the gate, the PMOS transistor undergoes an alternation of stress and recovery. Thus, the impact of NBTI is determined by the operation of the circuits, and varies with different applications, or diffident phases in one application. An early analytical study [6] showed that the NBTI progress is determined by the signal probability of the input signal being "0", temperature and supply voltage, but independent of how frequently the transistor switches between "0" and "1". As a

978-3-9810801-6-2/DATE10 © 2010 EDAA



(a) Three NBTI modes: Stress (left), Moderate Recovery (middle), Proactive Recovery (right)



(b) Proactive recovery circuits for SRAM cells [12]

Fig. 1. NBTI and its recovery.

result, having sustained recovery span is as equally effective as having interrupted recovery periods as long as their accumulated recovery cycles are the same.

A stronger and faster recovery mode for PMOS transistors, referred as *proactive recovery*, is shown in the last case of Fig. 1(a). It uses positive voltage bias, i.e., $V_{gs} = V_{dd}$ [8] and forces the gate input to be "1" and S terminal to be "0". The MOS transistor cannot enter the strong recovery mode in normal operation since (1) the S terminal of the PMOS is connected to physical V_{dd}; (2) even if S terminal can be driven to ground, the gate voltage is discharged through the preceding PMOS and cannot be kept at "1". Therefore the cell circuit needs to be revised properly to enable proactive recovery.

B. Proactive NBTI Recovery for L2 Caches

A modern microprocessor has many functional units using storage circuits e.g. caches, registers, buffers. The proactive recovery scheme proposed for L2 cache cells [12] takes advantage of the symmetric structure of a 6T SRAM cell (Fig. 1(b)). The two PMOS gates of the cell are complementary to each other, and the recovery for them is done one at a time to ensure stability. That is, when the V_s of one PMOS is forced to the ground, the V_s of the other PMOS is kept at V_{dd}. Which V_s should be pulled down depends on the value stored in the cell. For example, if the cell stores a "0" ($bl = 0, br = 1$), then it is appropriate to recover the PMOS on the right side of the cell because the left PMOS can guarantee the br to hold at "1" while the virtual V_{dd} for the right PMOS is driven to ground. Symmetrically, when the cell is storing a "1", it is proper to recover only the left PMOS.

Hence, the recovery design for L2 cache cells [12] requires a preceding write operation to align all the cells for recovery. That is, to recover the PMOS transistors on the right side, it writes 0s to all candidate cells before the recovery. Due to the destructive write operation, the recovery logic needs to back up the cache contents in a spare array prior to the write. These operations incur both latency and area overheads, and thus were applied only to L2 caches but not to storage cells within the processor core.

III. Proactive NBTI Recovery for Busy FUs

Within a processor core, many critical and busy functional units (FUs) such as RS (reservation stations), ROB (reorder buffers), and PR (physical registers) contain PMOS based storage cells. It is preferable to apply proactive NBTI recovery for these FUs for two reasons: (1) they are more vulnerable to NBTI-induced failure than other FUs since they are busy and hot in an out-of-order microprocessor, and NBTI effect becomes more severe with higher biased probability and higher temperature; (2) they are crucial FUs whose reliability degradation may directly fail the entire chip.

However it is challenging to perform proactive NBTI recovery within the processor core. First, these FUs are frequently used and thus do not have abundant idle time to perform NBTI recovery at the bank level, i.e the way to recover L2 cache banks in [12]. Second, there are no spare copies of FUs to store the old values. To overcome these difficulties, we propose a per-buffer-entry level proactive scheme that exploits the idle time of individual entries, and performs proactive recovery during their *inactive* cycles i.e. the contents are no longer used and can be safely discarded. We next analyze the recovery opportunities in these FUs, and present our hardware design in the next section.

A. FU Activities

Let us first analyze the idle and busy cycles of above FUs in different stages of execution: dispatch, issue, execute, complete and retire (commit).

Instructions are fetched and decoded sequentially. Prior to dispatching an instruction to the out-of-order engine, the hardware has to check resource availability based on three conditions:

1) is there a free PR to hold the output?
2) is there a free entry in the RS to hold the operand tags? and
3) is there a free entry in the ROB to hold related information?

If any condition is not satisfied, the instruction is stalled without allocating any entry in these FUs (PR, RS and ROB). Otherwise the instruction is dispatched, and each FU allocates a free entry for the instruction. Those entries turn from *idle* state to *busy* state.

In the issue and execute stages, all the allocated entries are busy, so recovery is forbidden. The transition from busy state to idle state is associated with the deallocation of the entries. In non-speculative execution, the RS entries are deallocated in the complete stage and the ROB entries are deallocated in the retire stage. A PR is freed only when all the following three conditions are satisfied:

1) it is not referenced by the register aliasing table (RAT);
2) there is no pending consumer (reader); and
3) there is no unresolved branch instruction.

In case of a mis-speculation, the entries in the RS and ROB on the mispredicted path are squashed and deallocated. Also, the newly allocated PRs associated with the squashed ROB entries are freed. And the RAT is recovered to the state where the misprediction begins. However, no PRs prior to the mispredicted branch are allowed to be freed because otherwise, their architectural states would be lost. This corresponds to the third condition of freeing a PR above.

In summary, when a RS/ROB/PR entry is allocated, it is in the busy state and recovery cannot be performed. Once it is deallocated, it becomes free, and the content can be discarded for recovery purpose – proactive recovery destroys the stored information. Therefore, we keep tracking the entry states and capture the idle cycles to perform proactive recovery. However, proactive recovery incurs delays when driving down and up the virtual V_{dd} of entries. Hence, the idle period of an entry should be sufficiently long to encompass effective recovery cycles.

B. Idle Cycle Analysis

Fig. 2 reports the percentage of time that RS/ROB/PR entries stay idle for SPEC2000 INT and FP benchmark programs. For each

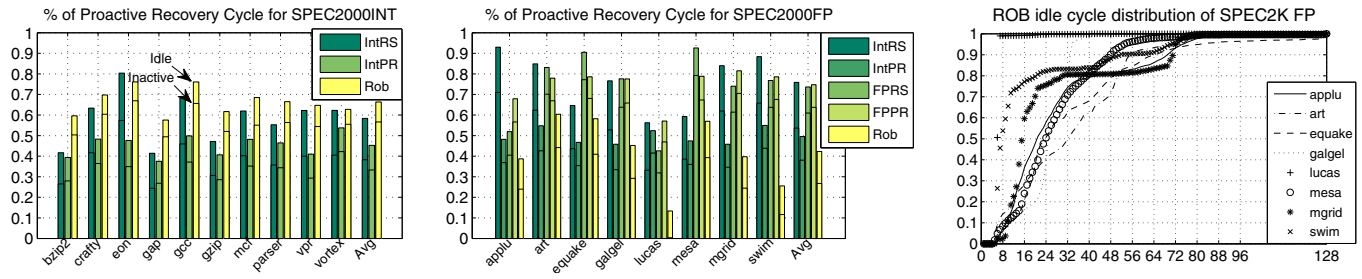


Fig. 2. Idle time percentages for SPEC2000 INT and FP benchmarks; Cumulative Distribution of idle durations for Rob under SPEC2000 FP benchmarks.

stacked bar in the figure, the top line (labeled as *idle*) indicates the idle cycle percentage; the lower line (labeled as *inactive*) indicates the portion that can be recovered — it will be discussed in Section III.C. Here we applied *round-robin* to allocate buffer entries in each FU. A round-robin allocation scheme favors reliability-aware designs as it balances the idles cycles over all entries. Our results showed that the entries in each FU have similar busy/idle cycle distribution and the average is reported in the figure.

The results illustrate that there are plenty of opportunities for NBTI recovery in those FUs. For example, for benchmark `applu`, the average idle time per entry accounts for over 90% of execution time for integer RS (IntRS). From the figure, the average percentages of idle time for integer RS, integer PR, floating point RS, floating point PR, ROB are: 76% (58% for SPEC2000 INT), 50% (45% for SPEC2000 INT), 74%, 75%, and 43% (66% for SPEC2000 INT). Our observation is consistent with previous work that shows 54%/69% of idle time for interger/FP PR [2], despite of different benchmarmks and processor configurations used in this paper. Notice that we utilize idle cycles at the *entry level* of those FUs, which gives us more opportunities than does at the entire FU level. Idle cycles of those entries come from under utilization of the out-of-order pipeline due to several reasons such as cache misses, mis-predictions, etc.

C. Recovery Opportunities with Latency Overhead

Since performing proactive NBTI recovery incurs extra delay i.e. it takes several cycles to change an entry to and from recovering mode, not all idle cycles can be exploited for proactive recovery. In particular, if the duration of an idle period is smaller than the transition overhead, it might be not beneficial to apply proactive recovery.

To study the recovery opportunities with transition overhead considered, we plotted the cumulative percentage distribution of idle periods for ROB in SPEC2000 FP benchmark programs in Fig. 2(right). Here the x axis is the duration of idle periods spanning from 1 to 128 cycles. We lump sum the duration beyond 128 cycles to 128, so all curves eventually reaches 1 at 128. In general, the slower a curve rises, the higher the percentage of longer idle durations. For example, ROB entries for *art* and *equake* tend to have longer idle durations than other programs, which corresponds to the highest average idle percentage shown in Fig.2(middle). A sharp rise of the curve indicates high percentages of the corresponding idle periods. For example, the *lucas* has a sharp rise for the idle period duration between 6 and 7 cycles, which indicates ∼50% of idle period durations are less than 6 cycles and almost all the rest are 7 cycles.

Having many short idle durations reduces the recovery opportunities. For example, if the overhead is 5 cycles in total, then it has no or negative benefit to proactively recover those idle periods that are less or equal to 5 cycles; and it has small benefits when recovering the idle periods slightly larger than 5 cycles. In Fig. 2, the lower portion (labeled as *inactive*) of each stacked bar indicates the percentage of idle cycles that can be exploited for recovery. Here we assume the transition overhead is 5 cycles as modeled in Section IV. As an example, for the ROB of `lucas`, there is little opportunity for proactive recovery, which gets explained by the sharp rise in Fig. 2(right). From the figure, assuming a fast per-entry based proactive recovery scheme can be designed, we found that a large amount of idle cycles can be exploited. On average out of the 43∼76% cycles identified as idle in aformentioned FUs, 27∼61% cycles can be exploited for proactive recovery.

In Fig. 2 we also see that RS and PR are busier than ROB when running SPEC2000 INT programs; and the ROB is the busiest when running SPEC2000 FP programs. This is because floating point instructions have longer latencies, and tend to stay in the ROB waiting for earlier long latency instructions to produce the results.

IV. The Hardware Design

To exploit the exposed recovery opportunities in the preceding section, we next present our hardware design (used as IP block in the EDA flow) details and then model its latency and area overheads.

To assist the discussion with proactive recovery, we distinguish three states for each entry in RS/ROB/PR — a *busy* state indicating the entry has been allocated; a *ready* state indicating the entry can be allocated and enter the busy state immediately; an *inactive* state indicating the entry is in recovering mode and has to switch to the ready state before being allocated again. The *ready* and *inactive* states correspond to the *idle* state when no NBTI recovery is considered. Switching from busy to inactive state (i.e. entering recovery), and from inactive to ready state (i.e. exiting recovery) incur the transition overhead. The transition between busy and ready states does not incur extra overhead.

Our proactive NBTI recovery is performed at per-entry level. When an entry is marked as *inactive*, the virtual power to this entry is driven to ground. To achieve maximal recovery, it is preferred to have more idle entries in the inactive state. However, inactive entries must exit recovery mode before being allocated to instructions. In order to avoid structural hazards in the dispatch stage, there should be enough ready entries available, and some inactive entries may have to switch to ready state in advance. This is especially necessary for multi-issue processors that allocate multiple RS/ROB/PR entries per cycle. Thus we need to design a control logic to achieve better trade-off between the attainable reliability improvement and the performance.

A. Proactive Per-entry Recovery with Automatic Bit Flipping

Our proactive NBTI recovery logic at the per entry level is similar to the one used in [12]. We enhanced the design with automatic bit flipping. As we discussed, [12] requires a write operation to align values in the storage cells before recovering them. This write is indeed not necessary and removing it helps reducing the overall transition overhead.

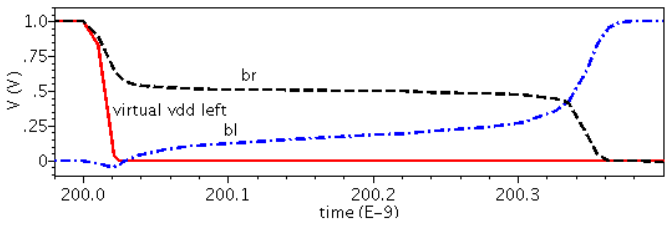


Fig. 3. Proactive NBTI recovery with automatic bit flipping.

We performed HSPICE simulation to study what happens if a cell stores a "0" and the virtual V_{dd} of its left PMOS is pulled down. The result in Fig. 3 shows that the cell value is flipped to "1" and kept stable from then on. For this reason, the left PMOS can enter the proactive recovery mode without value alignment. The transition overhead for automatic bit flipping is trivial — only $380ps$ was observed using the 45nm technology library [1]. While this operation destroys the contents, it is performed when the cell is in *inactive* state and thus there is no need to back up the old value.

B. The Control Logic for NBTI Recovery

As shown in Fig. 4, the control logic for enabling per-buffer-entry NBTI recovery consists of a power-down logic, a wake-up logic, and a ready entry counting logic.

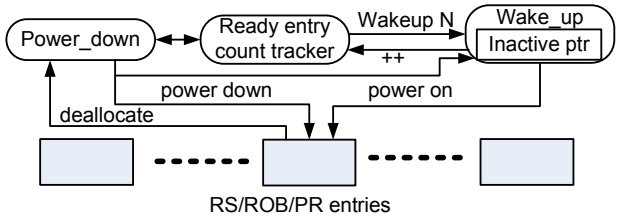


Fig. 4. Recovery control logic.

Power down logic. When a busy entry is to be deallocated, the power-down logic determines its next state which is either ready or inactive state. An inactive state automatically triggers proactive NBTI recovery to recover the corresponding entry.

By monitoring the status of each entry, the logic detects its deallocation point as early as possible such that the recovery benefits can be maximized. For RS and ROB, the logic monitors the busy bit associated with each entry. The RS entries are deallocated right after the execution stage, and the ROB entries are deallocated right after the commit stage. For the PR entries, the logic needs to check the reference bit, the consumer list and the speculation mode bit. A PR entry can be deallocated only when it is not referred by any RAT, i.e. there are no pending consumers or unresolved branches.

To determine the appropriate next state, the power-down logic strives to ensure enough ready entries in each FU such that structural hazards can be avoided and performance loss can be minimized. It tracks the number of ready entries, and turns the state of a to-be-deallocated busy entry to ready state if the number is below a threshold. Otherwise the logic turns the entry to inactive state. The threshold number of ready entries correlates with the issue width and the transition latency from inactive state to ready state. The reason for the latter correlation is that inactive entries are forced to switch to ready state in the case if there are inactive entries, and instructions get stalled due to insufficient ready entries. Since this transition costs 2 cycles (as discussed in Section IV.C), we found that preparing three times the issue-width of ready entries is sufficient and not too conservative. As an example, we maintain 12 ready entries in each FU for a 4-issue microprocessor.

An exception occurs when a mis-speculation is detected. At this time, the processor state is rolled back to the mis-predicted branch, and the corresponding RS and ROB entries are deallocated. These entries are highly likely to be re-allocated again along the correct branch path. Hence, they are turned to ready state directly, without additional checks. This decision favors performance, and also has trivial effect to recovery as the misprediction rate is low.

Wake-up logic. The power-down logic alone cannot maintain sufficient ready entries all the time since there are times when instructions do not commit for a long time due to a cache miss. We use the "wake-up" logic to dynamically monitor the number of ready entries. If the number of ready entries is below the threshold, the wake-up logic will locate several inactive entries and switch them to ready state. The wake-up logic and power-down logic work closely to balance the trade-off between attainable reliability and performance.

Ready entry counting logic. When a RS/ROB/PR entry is deallocated, the power-down logic gets notified. It then queries the *ready entry count tracker* to decide if it should apply the proactive recovery to the entry. If no action is taken, the entry is in ready state by default. The tracker monitors if there are sufficient number of ready entries in each cycle. If the repository drops below the threshold, a signal is sent to the wake-up logic to wake up necessary number of inactive entries. Since the entry allocation/deallocation runs in a circular queue manner, the wake-up logic keeps pointers to the first few inactive entries for ease of management. The pointers are updated by the power down signal (increment) and the power on signal (decrement). The *ready entry count tracker* maintains the current number of ready entries. The counter is updated when 1) a ready entry is allocated by the dispatcher (not shown); or 2) the power-down logic leaves a busy entry in the ready state; or 3) the wake-up logic powers on an entry.

C. Latency Overhead

The latency overhead includes the delay in deciding whether to apply proactive recovery and the delay of driving the virtual V_{dd} to ground/power. We synthesized the control logic for all RS/ROB/PR units using $45nm$ technology. The delays for making power down and up decisions are 2 and 1 cycles respectively.

To quantify the delay of driving the virtual V_{dd} to ground/power, we built the circuit of the register entry with 4 read ports and 2 write ports using 45nm technology library [1] in Cadence Virtuoso. With proper selection of the stages and sizes of the buffers, the rising ($0 \rightarrow 0.95\ V_{dd}$) and falling ($V_{dd} \rightarrow 0.01 V_{dd}$) time of virtual V_{dd} for a 64-bit register or ROB entry are 130ps and 130ps respectively (2 stage buffer). For a 256-bit RS entry [2], the falling and rising time are 150ps and 150ps respectively (3 stage buffer). That is, it takes 1 cycle for a 2Ghz core to drive the virtual V_{dd} to ground/power for any single entry in these FUs. The time to drive a single entry is much faster than that to drive the whole FU as it requires smaller diffusion capacitance on the gating path and has stronger drive current in NMOS.

In summary, it takes 3 and 2 cycles respectively for the transition of entering and exiting the proactive recovery mode.

D. Area Overhead

The area overhead of the per-buffer-entry proactive recovery logic includes the idle detection logic, the buffer to drive the virtual V_{dd}, and the center control logic.

The idle detection logic consists of simple AND gates based on the analysis in Section III-A, with the state bits from different entries as the input. The buffer logic incurs trivial area overhead due to the small load of a single entry. The storage cell in 45 nm technology

is $1.144\mu m^2$, extracted from CACTI [7]. The buffer area of the 64/256-bit entry is $0.408/1.632\mu m^2$, adding only 0.56% overhead to the design. As for wring, using the similar design of controling the wordline drivers in [15], the per-entry control is implemented without adding wires in SRAM arrays. Hence, only wiring the control signals from the control unit to every entry adds the area outside the SRAM array. Besides the logic, seperate virtual V_{dd} rails are needed, which often exist [12] or can be minimized with proper layout planning. The area of controller unit for all FUs is $0.12mm^2$ ($<<$1% for most out-of-order processors), and power is $2.677mW$, which is negligible for the whole chip.

V. Simulation and Results

A. Simulation Setup

We evaluated our proposed design on an out-of-order processor whose configurations are summarized in Fig. I. We used a cycle accurate simulator enhanced from Simplescalar Sim-Alpha [3] to study the recovery opportunity, the 0/1 input distribution at runtime, and the performance impact of our design. The original Sim-Alpha simulator used a unified data structure to model the timing effect of RS, ROB and PR for simplicity. To accurately characterize these FUs, we modeled them independently following the discussion in Section III-A. The ROB entries store the instruction order information. The RS entries track and resolve the operand dependency. The PR entries hold the source operands. We chose 10 integer (INT) and 8 floating-point (FP) programs from SPEC2000 benchmark suite. We skipped their warm up phases, and simulated one billion instructions for performance study.

TABLE I
Processor configurations.

Configuration	Parameter
ISA	Alpha 21264
Reservation station	60 FP and 60 Integer
Reorder buffer	128
Physical register	128 INT and 128 FP
Load store queue	32
Fetch queue	32
L1 I-cache	32 KB 4way
L1 D-cache	32 KB 4way
L2 unified cache	4MB 8way

Reliability model. In order to quantify the reliability improvement after applying our proposed proactive NBTI recovery, we modeled the MTTF (Mean Time to Failure) for each affected FU as follows.

In this paper the failures are limited to those due to NBTI induced V_{th} degradation. Since there are no ECC bits in busy FUs, one single bit failure is a hard defect and can not be corrected. We conservatively assume the first hard bit failure cause the failure of the chip. The failure model for storage cell bits, introduced in [9], includes read failure, write failure, hold and retention failure. [4] showed that read stability is the most critical one for NBTI degradation. Therefore we adopted read mode SNM (Static Noise Margin) degradation, which is closely related to the read stability, as the indicator of circuit failures in this paper. That is, a cell fails when its SNM is below a preset threshold. The SNM of a single SRAM cell is determined by the V_{th} deviation from nominal values of the transistors. The V_{th} deviation comes from both process variations and NBTI induced V_{th} degradation. The latter keeps enlarging the deviation until SNM reaches the threshold.

MTTF is computed in four steps. (1) Since the relationship of SNM and ΔV_{th} is non-linear, we used the interpolation method to find the critical ΔV_{th}. For different PMOS V_{th} settings, we performed HSPICE DC analysis to obtain voltage-transfer-characteristic (VTC) and then the corresponding SNM. Based on the discrete points, interpolation was used to calculate the critical ΔV_{th} that causes the cell failure. (2) We then calculated the margin for NBTI induced V_{th} shift for each cell i.e. the $\Delta V_{t,AC}$ in equation 1. Our model integrated the static process variation (using the model from VARIUS [10]) as well. (3) We computed the time-to-failure (TTF) of a cell by solving Eqn 1 [4] for t:

$$\Delta V_{t,AC} = K_{AC} \cdot t^n = \alpha(S, f) \cdot K_{DC} \cdot t^n \quad (1)$$

Here K_{DC} is the constant determined by technology. $\alpha(S, f)$ is the AC factor determined by the signal probability of the cell and operation frequency. $\alpha(S)$ is used due to frequency independence [6]. The signal probability of all bits are recorded in the micro-architecture simulation. The solved t represents the TTF of one cell. (4) We computed the FU level TTF using the minimum t (first failed bit) of all cells. MTTF is the mean TTF of multiple FU samples.

The function $\alpha(S)$ was derived using the detailed model in [14] with stress and recovery modes: positive interface traps (N_{it}), which is proportional to (ΔV_{th}), is determined by:

$$\text{Stress:} \quad N_{it} = \sqrt{K^2(t - t_0)^{\frac{1}{2}} + N_{it0}^2} \quad (2)$$

$$\text{Recovery:} \quad N_{it} = N_{it0}[1 - \sqrt{\eta(t - t_0)/t}] \quad (3)$$

where N_{it0} is the number of positive interface traps when the stress or recover mode begins at t_0. η and K are process related parameters. Different signal probabilities S result in different ratio of stress and recovery durations and final N_{it}. The normalized N_{it} over N_{it0} is the α determined by S. For our proactive recovery mechanism, based on the parameter extracted from the result in [8], η is scaled to represent the stronger recovery effect, resulting $\alpha'(S)$ which is different from $\alpha(S)$. For the same S, $\alpha'(S) < \alpha(S)$.

B. Results

MTTF improvement. Fig 5 compares the FU level MTTF when applying three different designs. The results for `Bal` and `Pro` are normalized to `Normal`, which is set at 1.0.

- The `Normal` design is the baseline that has natural recovery. That is, there is no special recovery treatment.
- The `Bal` design applies special values or inverted values to these FUs during idle time [2]. The goal is to balance 0/1 input distribution for each transistor.
- The `Pro` design is our proposed per-entry based proactive NBTI recovery scheme.

From Fig 5, `Bal` achieves at most 2.76× and on average 2.34× MTTF improvement over `Normal`. This MTTF improvement comes from balancing NBTI degradation of the complementary PMOS. Since most FUs exhibit abundant idle cycles, despite varying input 0/1 distribution for different cells, there is a large room to balance the distribution for complementary PMOS transistors. However, the V_{th} recovery is limited due to its moderate recovery strength in normal mode. From the figure, the MTTF improvement depends on the percentage of idle cycles. For example, the ROB MTTF for *lucas* has little improvement due to its limited idle cycles. We noticed that the MTTF improvements are flat for ROB under SPEC2000 INT and FPPReg under SPEC2000 FP. The reason is that there are sufficient large number of idle cycles for different programs, and `Bal` achieves near optimal input signal balancing for these FUs.

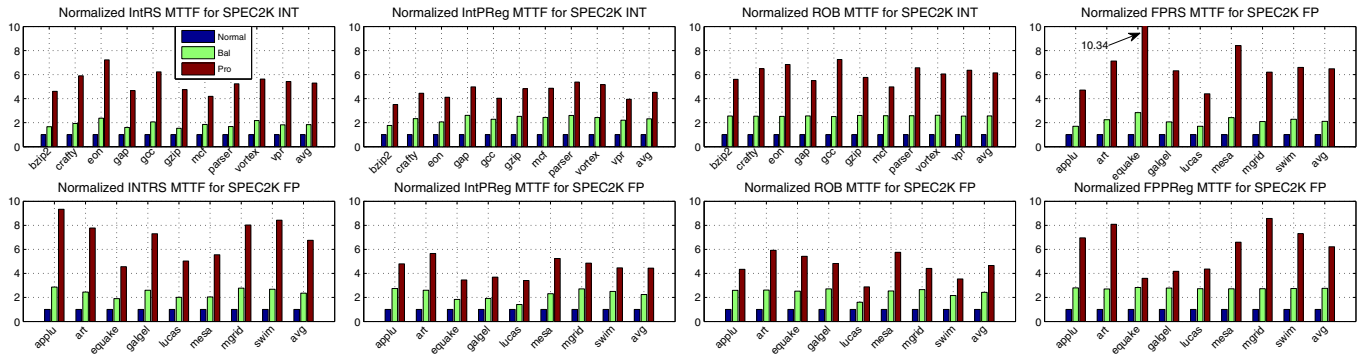


Fig. 5. MTTF improvement of BAL and PRO for FUs using SPEC2000.

`Pro` achieves at least 4.44× and on average 5.57× MTTF improvement, which is superior over `Bal`. Although only the inactive cycles (a portion of idle cycles) are utilized to apply recovery, the stronger strength under proactive recovery results in better MTTF improvements. Similar to `Bal` depending on idle cycles, `Pro` depends on the amount of inactive cycles. For example, for SPEC2000 FP benchmarks, the IntRS and FPRS have largest MTTF improvement. `Pro` sometimes achieves better MTTF at the cost of performance loss. For example, *lucas* gains MTTF improvement while it only has limited inactive cycles in ROB. It has a relatively large performance degradation comparing to others (shown in Fig. 6). `Pro` did not try to always balance the 0/1 signal distribution as it only does so during inactive cycles but not during busy and ready cycles.

TABLE II
MTTF IMPROVEMENT OF DIFFERENT FUS.[a]

Design	IntPreg	FPPReg	IntRS	FPRS	ROB
Bal	2.24/2.32	2.76/-	2.36/1.84	2.11/-	2.43/2.56
Pro	4.44/4.52	6.22/-	6.76/5.30	6.49/-	4.66/6.15

[a]For each entry X/Y, X and Y represent the averages of all FP and INTEGER programs respectively

IPC. Due to the transition delay to and from proactive NBTI recovery, the program performance degrades when insufficient ready entries are maintained in these FUs. Fig. 6 compares the IPCs with and without proactive NBTI-recovery. We observed on average 0.5% IPC slowdown when setting the threshold to be the triple of the pipeline issue width. The exception is a 3.7% slowdown for *lucas* in which case there is little recovery opportunity and putting entries into recovery state incurs extra overhead to switch back to ready state.

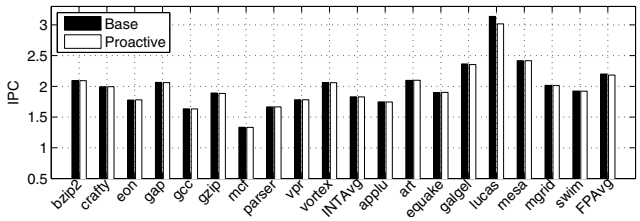


Fig. 6. Impact of proactive recovery on IPC.

VI. CONCLUSION

In this paper we identified the importance of applying proactive NBTI recovery for crucial and busy FUs (RS, ROB and PR) in modern microprocessors. We designed a per-entry based proactive NBTI recovery scheme to leverage the abundant idle opportunities of individual entries in those FUs. Compared to the baseline design with natural recovery, our scheme extends the FU lifetime by 5.57× on average and only incurs <1% area and performance overheads.

ACKNOWLEDGMENT

This work is supported in part by NSF grants: 0747242, 0641177, 0720595, and 0734339.

REFERENCES

[1] F. 45nm. A variation-aware design kit for 45nm. 'http://avatar.ecen.okstate.edu/projects/scells/OSUFreePDK45/indexframe.html.
[2] J. Abella, X. Vera, and A. Gonzalez. Penelope: The nbti-aware processor. In *Int'l Symp. on Microarchitecture*, pages 85–96, 2007.
[3] R. Desikan, D. Burger, S. W. Keckler, and T. Austin. Sim-alpha: a validated, execution-driven alpha 21264 simulator. In *Technical report TR-01-23*, 2001.
[4] K. Kang, H. Kufluoglu, K. Roy, and M. A. Alam. Impact of Negative-Bias temperature instability in nanoscale SRAM array: Modeling and analysis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(10):1770–1781, 2007.
[5] S. Kumar, K. Kim, and S. Sapatnekar. Impact of NBTI on SRAM read stability and design for reliability. In *Quality Electronic Design, 2006. ISQED '06. 7th International Symposium on*, pages 6 pp.–218, 2006.
[6] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. An analytical model for negative bias temperature instability. In *ICCAD'06*, pages 493–496. ACM, 2006.
[7] H. Labs. Cacti. 'http://www.hpl.hp.com/research/cacti/.
[8] S. Mahapatra. Mechanism of negative bias temperature instability in cmos devices: Degradation recovery and impact of nitrogen. In *Proc. of Int'l Electron Devices Meeting*, pages 105–108, 2004.
[9] S. Mukhopadhyay, H. Mahmoodi, and K. Roy. Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(12):1859–1880, 2005.
[10] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. Varius: A model of process variation and resulting timing errors for microarchitects. *IEEE Trans. on Semiconductor Manufacturing*, 21(1):3–13, February 2008.
[11] C. Schlunder, R. Brederlow, B. Ankele, A. Lill, K. Goser, and R. Thewes. On the degradation of p-mosfets in analog and rf circuits under inhomoheneous negative bias temperature stress. In *Proceedings of IRPS*, 2003.
[12] J. Shin, V. Zyuban, P. Bose, and T. M. Pinkston. A proactive wearout recovery apporoach for exploting microarchitectural redundancy to extend cache sram lifetime. In *Int'l Symp. on Computer Architecture*, pages 353–362, 2008.
[13] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. Exploiting structural duplication for lifetime reliability enhancement. In *Proc. of Int'l Symp. Computer Architecture*, pages 1–12, 2005.
[14] R. Vattikonda, W. Wang, and Y. Cao. Modeling and minimization of pmos nbti effect for robust nanometer design. In *Proc. of Design Automation Conf.*, pages 1047–1952, 2006.
[15] Y. Wang, U. Bhattacharya, F. Hamzaoglu, P. Kolar, Y. Ng, L. Wei, Y. Zhang, K. Zhang, and M. Bohr. A 4.0 ghz 291mb voltage-scalable sram design in 32nm high-k metal-gate cmos with integrated power management. In *Proc of ISSCC 2009*. ISSCC, IEEE, 2009.
[16] S. Zafar. Statistical mechanics based model for negative bias temperature instability induced degradation. *Applied Physics*, 97(10), May 2005.