

# Enhancing NBTI Recovery in SRAM Arrays Through Recovery Boosting

Taniya Siddiqua and Sudhanva Gurumurthi, *Senior Member, IEEE*

**Abstract**—Negative bias temperature instability (NBTI) is an important lifetime reliability problem in microprocessors. SRAM-based structures within the processor are especially susceptible to NBTI since one of the pMOS devices in the memory cell always has an input of “0”. Previously proposed recovery techniques for SRAM cells aim to balance the degradation of the two pMOS devices by attempting to keep their inputs at a logic “0” exactly 50% of the time. However, one of the devices is always in the negative bias condition at any given time. In this paper, we propose a technique called Recovery Boosting that allows both pMOS devices in the memory cell to be put into the recovery mode by slightly modifying to the design of conventional SRAM cells. We evaluate the circuit-level design of a physical register file and an issue queue that use such cells through SPICE-level simulations. We then conduct an architecture-level evaluation of the performance and reliability of using area-neutral designs of these two structures. We show that Recovery Boosting provides significant improvement in the static noise margins of the register file and issue queue while having very little impact on power consumption and performance.

**Index Terms**—Negative bias temperature instability (NBTI), static random access memory (SRAM).

## I. INTRODUCTION

**R**ELIABILITY is one of the biggest challenges facing the microprocessor industry today. With continued technology scaling, processors are becoming increasingly susceptible to hard errors. Hard errors are permanent faults that occur due to the wearing out of hardware structures over time. These failures occur partly due to design-time factors such as process parameters and wafer packaging, as well as runtime factors such as the utilization of the hardware resources and the operating temperature. It is important to ensure that the reliability of the microarchitectural structures in the processor is maximized so that one can make use all the available hardware resources effectively over the entire service life of the chip.

One important hard error phenomenon is negative bias temperature instability (NBTI), which affects the lifetime of pMOS transistors. NBTI occurs when a negative bias (i.e., a logic input of “0”) is applied at the gate of a pMOS transistor. The negative bias can lead to the generation of interface traps at the Si/SiO<sub>2</sub>

Manuscript received August 16, 2010; revised November 10, 2010; accepted January 13, 2011. This work was supported in part by the National Science Foundation under Grant 0627527 and by Intel Corporation.

The authors are with the Department of Computer Science, University of Virginia, Charlottesville, VA 22903 USA (e-mail: taniya@cs.virginia.edu; gurumurthi@cs.virginia.edu).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TVLSI.2011.2109973

interface, which cause an increase in the threshold voltage of the device. This increase in the threshold voltage degrades the speed of the device and reduces the noise margin of the circuit, eventually causing the circuit to fail [11], [14]. One interesting aspect of NBTI is that some of the interface traps can be eliminated by applying a logic input of “1” at the gate of the pMOS device. This puts the device into what is known as the recovery mode, which has a “self-healing” effect on the device [1].

Memory arrays that use static random access memory (SRAM) cells are especially susceptible to NBTI. SRAM cells consist of cross-coupled inverters that contain pMOS devices. Since each memory cell stores either a “0” or a “1” at all times, one of the pMOS devices in each cell always has a logic input of “0.” Since modern processor cores are composed of several critical SRAM-based structures, such as the register file and the issue queue, it is important to mitigate the impact of NBTI on these structures to maximize their lifetimes. Previous work on applying recovery techniques to SRAM structures aim to balance the degradation of the two pMOS devices in a memory cell by attempting to keep the inputs to each device at a logic input of “0” exactly 50% of the time [1], [11], [19]. However, one of the devices is always in the negative bias condition at any given time. In this paper, we propose a novel technique called *Recovery Boosting* that allows *both* pMOS devices in the memory cell to be put into the recovery mode. The basic idea is to raise the ground voltage and the bitlines to  $V_{dd}$  when the cell does not contain valid data.

The main contributions of this paper are given here.

- We describe how SRAM cells can be modified to support recovery boosting and discuss several circuit and microarchitecture-level design considerations when using such cells to build SRAM arrays.
- We present the circuit-level design of two large SRAM arrays in a four-wide issue processor core—the physical register file and the issue queue—that use the modified cells to provide recovery boosting. We verify the functionality of these designs and quantify their area and power consumption through SPICE-level simulation using the Cadence Virtuoso Spectre Circuit Simulator<sup>1</sup> for the 32-nm process technology. We show that the modified SRAM structures impose only a 3%–4% area overhead over the baseline non-recovery boost designs and that their maximum power consumption is less than 2% over the baseline.
- We then evaluate the performance and reliability of area-neutral designs of these modified structures at the architecture-level via execution-driven simulation using the M5

<sup>1</sup>Cadence Virtuoso Spectre Circuit Simulator. [Online]. Available: [http://www.cadence.com/products/cic/spectre\\_circuit/](http://www.cadence.com/products/cic/spectre_circuit/)

simulator [3] and the SPEC CPU2000 benchmark suite<sup>2</sup> in nominal operating condition. We show that recovery boosting provides a 56% improvement in the static noise margin of the register file cells and a 48% improvement for the issue queue across the benchmark suite while having a negligible impact on performance.

The remainder of this paper is organized as follows. Section II presents an overview of NBTI. Section III discusses the related work on NBTI mitigation techniques. Section IV discusses the recovery boosting technique and the circuit-level design, and evaluation of the register file and issue queue are given in Section V. The experimental methodology used for the architecture-level evaluation is given in Section VI, and the corresponding results in Section VII. Section VIII concludes this paper.

## II. OVERVIEW OF NBTI

When silicon is oxidized, most of the Si atoms at the surface of the wafer bond with oxygen while a few atoms bond with hydrogen. When a negative bias (i.e., a logic input of “0”) is applied at the gate of a pMOS transistor ( $V_{gs} = -V_{dd}$ ), the relatively weak Si-H bonds get disassociated, leading to the generation of interface traps at the Si/SiO<sub>2</sub> interface. These interface traps cause the threshold voltage ( $V_t$ ) of the pMOS transistor to increase, which in turn degrades the speed of the device and the noise margin of the circuit, eventually causing the circuit to fail [11]. A detailed discussion on the physics of interface trap generation due to NBTI is given in [14].

The period of time when the pMOS transistor is negatively biased is known as the *stress phase* or *stress mode*. The increase in  $V_t$  due to stress is given by the equation [23]

$$\Delta V_{ts} = \left( \frac{qt_{ox}}{e_{ox}} \right)^{3/2} \cdot K_1 \cdot \sqrt{C_{ox}(V_{gs} - V_t)} \cdot e^{-E_a/4kT + 2(V_{gs} - V_t)/t_{ox}E_{01}} \cdot T_0^{-0.25} \cdot t_{stress}^{0.25}$$

where  $t_{stress}$  is the time under stress,  $t_{ox}$  is the oxide thickness, and  $C_{ox}$  is the gate capacitance per unit area.  $K_1$ ,  $E_a$ ,  $T_0$ ,  $E_{01}$  and  $k$  are constants equal to  $7.5 C^{-0.5} nm^{-2.5}$ ,  $0.49 eV$ ,  $10^{-8} s/nm^2$ ,  $0.08 V/nm$ , and  $8.6174 \times 10^{-5} eV/K$ , respectively.

When a logic input of “1” is applied at the gate ( $V_{gs} = 0$ ), the transistor turns off eliminating some of the interface traps. This is known as the *recovery phase* or *recovery mode*. The final increase in  $V_t$  after considering both the stress and recovery phases is given by [23]

$$\Delta V_t = \Delta V_{ts} \cdot \left( 1 - \frac{2\xi_1 t_{ox} + \sqrt{\xi_2 e^{-E_a/kT} T_0 t_{rec}}}{(1 + \delta)t_{ox} + \sqrt{e^{-E_a/kT} (t_{stress} + t_{rec})}} \right)$$

where  $t_{rec}$  is the recovery time, and  $\xi_2$ ,  $\xi_1$ , and  $\delta$  are constants equal to 0.5, 0.9, and 0.5 respectively.

From the equations, one can observe several options for reducing the impact of NBTI. We can see that NBTI is affected by temperature, the stress and recovery times and the difference between  $V_{gs}$  and  $V_t$ . Reducing  $V_{gs}$ , temperature, the stress time and

increasing  $V_t$  can reduce the stress on the transistors whereas longer recovery times enhance the recovery process. However, using a lower  $V_{gs}$  (which also provides a reduction in temperature) can reduce performance. Similarly, although a higher  $V_t$  device is likely to be more resilient against NBTI, such devices are slower than their lower  $V_t$  counterparts. In this paper, we tackle the NBTI problem by increasing the recovery time.

## III. RELATED WORK

There are two basic approaches to mitigating NBTI: 1) reduce the stress on the pMOS transistors and 2) enhance the recovery process. Stress reduction techniques aim to reduce the aging rate by controlling  $V_{dd}$ ,  $V_t$ , and temperature, whereas recovery enhancement techniques aim to increase the recovery time for the pMOS devices. One could implement these techniques at a coarse granularity (e.g., for entire cores) [7], [21], [22] or for individual structures within the core [1], [9], [11], [19]. Recovery boosting is a recovery-enhancement technique for SRAM structures.

1) *Stress Reduction Techniques*: Srinivasan *et al.* [21] estimate the MTTF due to aging mechanisms at runtime based on the operating conditions and use dynamic reliability management (DRM) to stay within the reliability budget. Tiwari and Torrellas propose a technique called “Facelift” [22] to hide the effects of aging through temperature-based job-scheduling to individual cores of a multicore processor, in conjunction with  $V_{dd}$  and  $V_t$  control. Feng *et al.* [7] propose to use on-chip reliability sensors to guide job-scheduling decisions on a multicore processor. The design of such on-chip NBTI sensors are discussed in [5]. The use of stress reduction techniques is orthogonal to the use of recovery enhancement.

2) *Recovery-Enhancement Techniques*: Abella *et al.* [1] propose to feed specific bit patterns into the devices to increase the recovery time for pMOS transistors in logic structures (e.g., adders) during idle periods and balance the degradation of the pMOS devices in SRAM-based memory structures when they hold invalid data. Kumar *et al.* [11] propose a similar technique to periodically flip the contents of SRAM cells to balance the wear on the pMOS transistors. Shin *et al.* [19] propose a recovery enhancement technique for caches where SRAM cells are proactively put into the recovery mode via the use of a spare memory array. They reverse bias ( $V_{gs} = V_{dd}$ ) the pMOS devices to put them into a deep recovery state. When an array is put into the recovery mode, the pMOS devices in one of the inverters in all of the cells are put into the recovery mode followed by those in the other inverter and this recurring pattern is continued throughout the recovery period for the array. All of these recovery enhancement techniques aim to balance the degradation of the two pMOS devices in the memory cell by attempting to keep the inputs to each device at a logic input of “0” (i.e., negative-bias) exactly 50% of the time.

A recently issued U.S. patent describes an idea similar to one of our proposed recovery-boosting technique (coarse-grained) which is discussed in Section IV [4]. However, the idea in the patent has the following drawback: it takes multiple processor clock cycles to put both PMOSs into recovery and therefore cannot be used for high-speed SRAM arrays. Moreover, the patent does not provide any analysis of the impact (qualitatively

<sup>2</sup>CPU2000. [Online]. Available: <http://www.spec.org/cpu2000/>

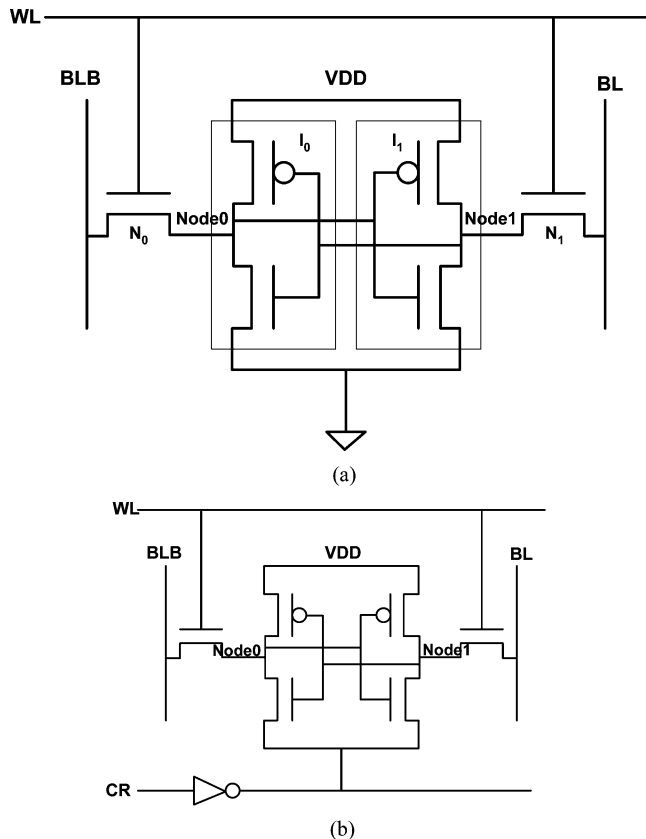


Fig. 1. SRAM cell design for recovery boosting. (a) Conventional 6T SRAM cell. (b) Modified SRAM cell that supports recovery boosting.

nor quantitatively) of using their technique to design microarchitectural structures.

#### IV. BASICS OF RECOVERY BOOSTING

Before we discuss recovery boosting, we first review the design and operation of a conventional 6-transistor (6T) SRAM cell. The design of the 6T cell is given in Fig. 1(a). The cell is composed of a wordline (WL), a pair of bitlines (BL, BLB), two cross-coupled inverters ( $I_0, I_1$ ), and two access transistors ( $N_0, N_1$ ). The cross-coupled inverters store one bit of data. There are three basic operations that one can perform on this SRAM cell: read, write and hold. To read and write data, the cell is selected by raising WL to high. This activates the access transistors and connects the inverters in the cell to the bitlines. During a read operation, both bitlines are first precharged high. Based on the data stored in the cell, one of the bitlines is discharged. This change is detected by a sense amplifier (which is not part of the cell) to determine the value stored in the cell. During a write operation, one of the bitlines is raised high and the other is lowered depending on the value to be written to the cell. When the cell is not selected ( $WL = 0$ ) for read or write, it is expected to hold the data stored in it and is said to operate in the hold mode.

Since the SRAM cell has cross-coupled inverters, each inverter charges the gate of the pMOS or nMOS device of the other inverter. Therefore, at any given time, one pMOS device will always be in the stress mode. The goal of recovery enhancement is to put the pMOS devices into the recovery mode by

TABLE I  
MODIFIED SRAM CELL OPERATION

CR	WL	BL	BLB	Node0	Node1	Operation
1	0	X	X	0/1	1/0	Hold
1	1	1	1	0/1	1/0	Read
1	1	1	0	0	1	Write '1'
1	1	0	1	1	0	Write '0'
0	X	X	X	1	1	Recovery Boost

feeding input values to the cell that will transition them into that mode. However, due to the cross-coupled nature of the inverters, only one of the pMOS devices can be put into the recovery mode. Therefore, previously proposed recovery-enhancement techniques attempt to balance the wearout of the two pMOS devices by putting each pMOS into the recovery mode 50% of the time by feeding appropriate input values [1], [11], [19]. We propose a 6T SRAM cell design shown in Fig. 1(b) which is capable of normal operations (read, write, and hold) as well as providing an NBTI recovery mode (when the cell does not contain valid data) that we call the *recovery boost mode* where both pMOS devices within the cell undergo recovery at the same time. We refer to the period when the cell does not contain valid data that is never used by any other microarchitectural structure in the processor as “invalid period.”

The basic idea behind recovery boosting is to raise the node voltages (Node0 and Node1 in Fig. 1) of a memory cell in order to put both pMOS devices into the recovery mode. This can be achieved by raising the ground voltage to the nominal voltage through an external control signal. The modified SRAM cell has the ground connected to the output of an inverter, as shown in Fig. 1(b). CR is the control signal to switch between the recovery boost mode and the normal operating mode. During the normal operating mode, CR has a value of “1” ( $V_{dd}$ ), which in turn connects the ground of the SRAM cell to a value of “0.” With this connection, the SRAM cell can perform normal read, write, and hold operations. To apply recovery boosting, CR has to be changed to a “0” in order to raise the ground voltage of the SRAM cell to  $V_{dd}$ . This circuit configuration puts both pMOS devices in the SRAM cell into the recovery mode. A cell can be put into the recovery boost mode regardless of whether its wordline (WL) is high or low. Unlike read and write operations on a cell, putting a cell into the recovery boost mode does not require an access to its wordline. The operations of the modified SRAM cell are shown in Table I.

However, the drawback of this approach is that it can take a long time to raise both the node voltages to  $V_{dd}$  in a high-performance processor that operates at a high clock frequency. This is illustrated in Fig. 2, which presents the achieved pMOS gate voltages of a bitcell over time due to recovery boosting. The simulation is performed using the Cadence Virtuoso Spectre circuit simulator for the 32-nm process using the Predictive Technology Model.<sup>3</sup> The operating temperature is 90 °C, which is the average temperature in which the high-performance processors operate [12]. We use this temperature value throughout the paper for all the experiments. We can observe that this approach achieves the desired gate voltage ( $V_{dd}$ ) within 3.33 ns. For a processor which operates at 3-GHz frequency, it will take ten cycles

<sup>3</sup>Predictive Technology Model. [Online]. Available: <http://www.eas.asu.edu/~ptm/>

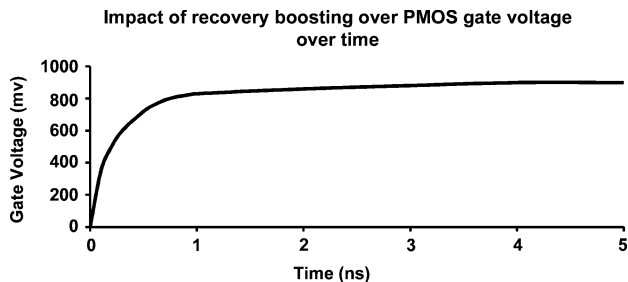


Fig. 2. PMOS gate voltages of an SRAM bitcell due to recovery boosting using the modified SRAM cell shown in Fig. 1(b) ( $V_{dd} = 0.9$  V,  $T = 90^\circ$  C).

to switch to the recovery boost mode. Similarly, it takes around ten cycles to go back to the normal operating mode from the recovery boost mode. However, our goal is to be able to switch between the recovery boost mode and the normal operating mode within a single cycle which is critical for a high-speed SRAM structure, such as the issue queue, where instructions need to be woken up and selected within a single clock cycle, in order to expedite the execution of dependent instructions. As mentioned before, recovery boost mode is applied when an entry of the structure holds data that is considered “invalid” at the architecture-level. Entries in the high-speed structures change their status between valid and invalid very frequently. For example, we find from architecture simulations that an issue queue entry stays invalid for about 50 cycles before it changes its status to valid. In such scenario, the cell shown in Fig. 1(b) will take 20 cycles of the 50 cycles (40% of the invalid period) to shift between modes, given that shifting to the normal operating mode takes place during the end of the invalid period. Thus, only 30 cycles could be utilized for the recovery process. On the other hand, if extra cycles are allocated to shift to the normal operating mode after the invalid period, that would have negative consequences on the processor performance. Therefore, single-cycle switching is required for the high-speed structures in the processor for the maximum utilization of the invalid states for the recovery process without any performance loss. Such single-cycle switching can be achieved by raising the bitlines along with the ground voltage to  $V_{dd}$ . There are various ways of incorporating such cells into SRAM arrays, which we will discuss shortly.

Recovery boosting can be provided at a fine granularity, such as for individual entries/rows of a memory array, or at a coarser granularity, such as for an entire array. We now discuss how the modified high-speed recovery boosting SRAM cells can be used in each of these scenarios and then discuss additional microarchitectural issues related to implementing recovery boosting.

#### A. Fine-Grained Recovery Boosting

In the normal operating mode, the state of the bitlines change during read and write operations. Since a pair of bitlines is shared by all the memory cells in a given column in the array, even those memory cells that are not being read from or written to will have the voltage on their bitlines changing. In an ordinary SRAM array, these bitline transitions do not affect the normal operation of the cells. However, in order to perform recovery boosting of a memory cell, both bitlines of the cell need to be

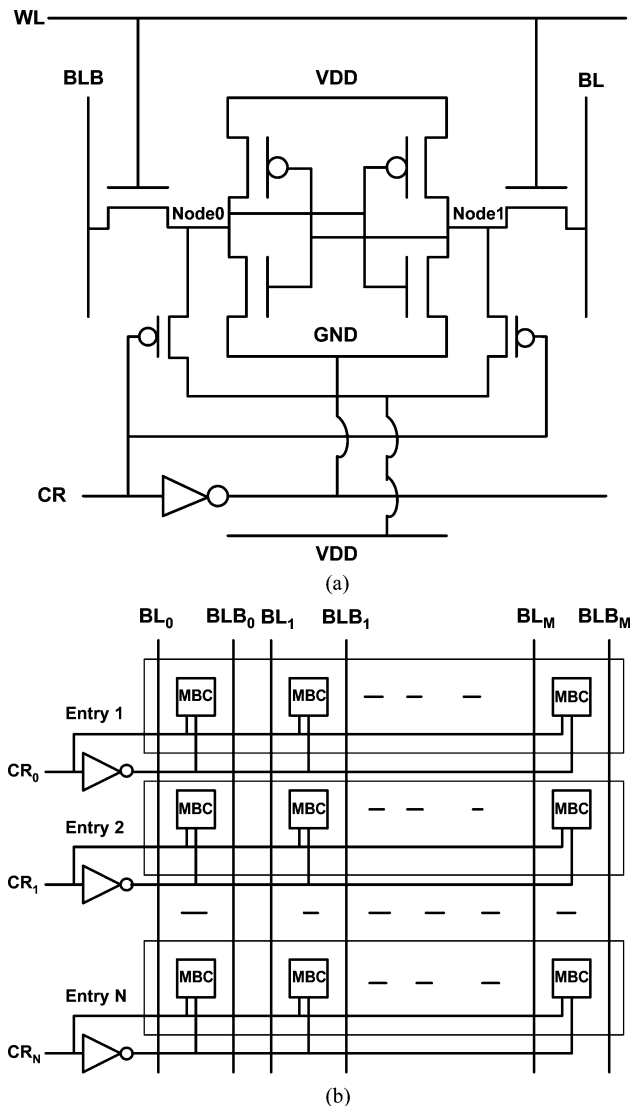


Fig. 3. SRAM array for fine-grained recovery boosting. (a) Modified SRAM cell with connection to the  $V_{dd}$  rail of an adjoining row. (b) SRAM array with modified cell ( $N$  entries,  $M$ -wide).

raised to  $V_{dd}$ . Therefore, we need to be able to isolate the bitlines of the memory cells that are in the recovery boost mode from the bitlines that are used for accessing other cells in the array. To provide this isolation, we extend the memory cell in Fig. 1(b) with connections to the  $V_{dd}$  rail of an adjoining row or column via two pMOS access devices. The design of the modified SRAM cell is shown in Fig. 3(a) and an SRAM array that uses this cell for controlling individual entries to operate either in normal or recovery boost mode is shown in Fig. 3(b).

In the memory cell design given in Fig. 3(a), the CR signal serves the same purpose as before. When a value of “0” is input to the CR line to transition the cell into the recovery boost mode, in addition to raising the ground voltage, the two extra pMOS devices connected to the  $V_{dd}$  rail are also turned on. Therefore, by raising the ground and connecting the bitcell to  $V_{dd}$ , the cell can be transitioned into the recovery boost mode without affecting cells in other rows of the array.

We make the extra pMOS devices resilient against NBTI by using high- $V_t$  transistors. Although high- $V_t$  devices are slower,

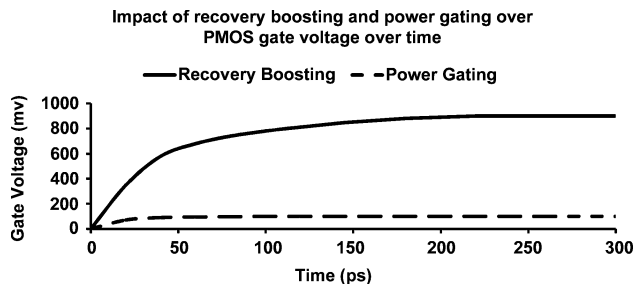


Fig. 4. PMOS gate voltages of an SRAM bitcell due to recovery boosting and power gating ( $V_{dd} = 0.9$  V,  $T = 90$  °C).

these devices are used only when transitioning the cell into the recovery boost mode and not when transitioning to the normal operating mode. Therefore, these devices do not impact performance but may delay the transition into the recovery boost mode. Moreover, since these devices do not lie on the performance critical path, they are sized so as to minimize the overall area. However, the pMOS devices do consume leakage power. We quantify the power consumption in Section V.

### B. Coarse-Grained Recovery Boosting

In this approach, we use the SRAM cell design shown in Fig. 1(b) instead of the one for fine-grained control. Here, a single control signal puts the entire array into the recovery boost mode. The control signal CR with a value of “0” raises the ground connection of each entry to  $V_{dd}$ . In this design, connections to the  $V_{dd}$  rail via the pMOS devices are not required. Instead, we merely need to raise all the bitlines in the array to  $V_{dd}$  to transition all the cells in the array to the recovery boost mode.

1) *Tradeoffs Between Fine-Grained and Coarse-Grained Recovery Boosting*: Going in for the fine-grained approach entails an area overhead of having two additional pMOS devices for each memory cell which can be prohibitive for large SRAM arrays such as caches. On the other hand, the fine-grained approach provides single-cycle switching with greater flexibility in managing NBTI by exploiting the usage characteristics of individual entries in the structure. In this paper, we evaluate the use of recovery boosting for the register file and issue queue. Due to the relatively small size of these structures (compared with caches), we use the fine-grained approach.

### C. Other Issues

1) *Difference Between Recovery Boosting and Power Gating*: Similar to recovery boosting, power gating also involves a small change to the design of the SRAM cell and can also be used to combat NBTI [24]. As shown in Fig. 4, we can observe that recovery boosting achieves the desired gate voltage ( $V_{dd}$ ) within a very short interval of time (195 ps), whereas power gating achieves only about 11% of  $V_{dd}$ . Power gating requires several thousand nanoseconds to reach  $V_{dd}$  to provide recovery to the SRAM bitcell. Therefore, it can be used as a stress reduction approach for the high-speed structures since the duration of the invalid phases of these structures tend to be smaller than thousand nanoseconds. When a memory cell stores valid data, neither recovery boosting nor power gating can be applied and the pMOS devices in the cells will be stressed in a similar way.

However, when the memory cell is idle and the data in the cell is no longer needed, it would be more beneficial to take advantage of recovery boosting.

2) *Impact of Process Variation on Correct Functionality*: In deep submicron technologies, intra-die process variation is an important issue. Different transistor parameters, including  $V_t$ , are affected by process variation and can impact circuit delay characteristics.  $V_t$  is affected by variations in the device geometry, random dopant number fluctuations and mobile charges in the gate oxide [20]. Process variation will affect the delay characteristics of the 6T SRAM cell inherent in both the original and modified bitcells in a similar way. Process variation can also impact the  $V_t$  of the two  $V_{dd}$ -rail access transistors and the devices in the inverter connected to the CR line. If the  $V_t$  of the  $V_{dd}$  rail transistors is high, then transitioning the bitcell to the recovery boost mode may be slower. This could reduce the amount of time for which we can apply recovery boosting but will not affect the correctness of the SRAM cell operation. On the other hand, if their  $V_t$  is lower, the bitcell will transition into the recovery boost mode faster and the access devices will consume higher leakage power but will again not affect correctness. A delay in the pMOS device of the CR line inverter will again merely slow the transition into the recovery boost mode. However, a delay in the nMOS device in the inverter could affect the speed at which the cell transitions out of the recovery boost mode and into a normal operating mode, which can affect correctness. In order to handle this situation, we need to set the clock frequency such that this delay can be accommodated within a single cycle.

3) *Recovery Boosting Does Not Exacerbate PBTI*: Putting the pMOS devices into the recovery mode does not increase the stress on the nMOS devices in the memory cell. Stresses on the nMOS devices can lead to a phenomenon that is similar to NBTI called positive bias temperature instability (PBTI), which occurs when a positive bias ( $V_{gs} = V_{dd}$ ) is applied to the nMOS device. As with NBTI, PBTI also generates interface traps and increases the threshold voltage. PBTI is expected to become more important in future deep submicrometer technologies [15]. While in the recovery boost mode, both the ground and node voltages of the cell are raised to  $V_{dd}$ . Consequently, the  $V_{gs}$  of the nMOS devices in the inverters becomes zero and therefore these nMOS devices do not experience any positive bias. The access transistors are also not accessed during the recovery boost mode. Therefore, recovery boosting does not exacerbate PBTI on the nMOS devices in the memory cell (and may in fact provide PBTI recovery [15]).

## V. DESIGNING MICROARCHITECTURAL STRUCTURES THAT SUPPORT RECOVERY BOOSTING

Having discussed the basics of recovery boosting, we now turn our attention to designing SRAM-based microarchitectural structures that use this technique to provide resilience against NBTI. Here, we present and evaluate the circuit-level design of two large SRAM-based structures within a four-wide issue processor core, namely, the physical register file and the issue queue, which we modify to support recovery boosting. We study the design of a 128-entry multiported physical register file with eight read-ports, four write-ports, and 64-b entries. The issue

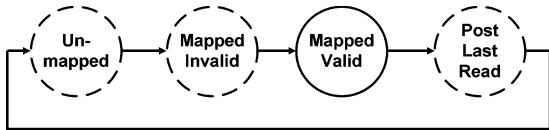


Fig. 5. Register states. The candidate states for recovery boosting are shown in dashed circles.

queue uses a non-data-captured design [17] and consists of 64 entries with four read-ports, four write-ports and 65 b per entry. The choice for the entry size is based on the issue queue descriptions given in [1].

#### A. Physical Register File

Superscalar processors attempt to exploit instruction-level parallelism (ILP) by fetching, decoding, executing, and retiring multiple instructions each clock cycle. In order to eliminate false dependences and support a large number of in-flight instructions, superscalar processors make use of register renaming. There are a number of microarchitectural options for implementing register renaming. In this paper, we model a microarchitecture that uses a separate architected register file and a physical register file. Instructions whose source operand values are to be supplied by a physical register have their architected source register mapped to the appropriate physical register during renaming. These mappings from architected registers to physical registers are maintained in a register alias table (RAT). The physical register is returned to the free list of registers when the next instruction that writes to the same architected register commits.

A physical register goes through a sequence of four states, shown in Fig. 5: 1) it is not mapped to any producer instruction and is free (*Unmapped*); 2) it is mapped to an instruction but it has not yet been written into by that instruction (*Mapped-Invalid*); 3) it holds a valid value that has been written to it (*Mapped-Valid*); and 4) it holds a valid value but the value is not read by any instruction before it is released to the pool of free registers (*Post Last-Read*). Once the register completes the *Post Last-Read* state, the register returns to the *Unmapped* state and remains in that state till the register-renaming logic chooses it again for a mapping.

There are three candidate states that one could use for recovery boosting: *Unmapped*, *Mapped-Invalid* and *Post Last-Read*. When a physical register is in the *Unmapped* and *Mapped-Invalid* states, it does not hold valid data and therefore we can put its cells into the recovery boost mode without affecting architectural correctness. The cells will need to be transitioned into the normal operating mode when moving from the *Mapped-Invalid* to the *Mapped-Valid* state, which occurs when the producer instruction has completed its execution and forwards the value to the register file. *Post Last-Read* is a more complex situation. Several cycles may elapse between the last time that the physical register is read and when the register is released. This time period could potentially be exploited for recovery boosting but it is challenging to precisely determine when the last-use of a register is complete. Although there have been proposals to exploit this time window to speculatively release registers early for the sake of performance and power

Current State	unmapped bit	completed bit	Control Signal (CR)	Switch to Recovery Boost?
Mapped/invalid	0	0	0	Yes
Mapped/valid	0	1	1	No
Unmapped	1	0	0	Yes
Unmapped	1	1	0	Yes

Fig. 6. Control signal truth-table for a register.

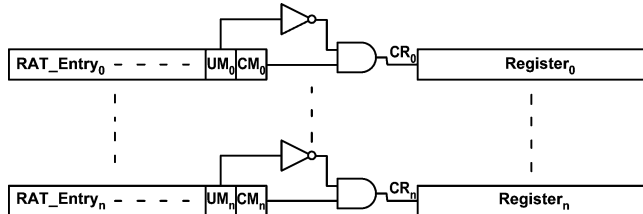


Fig. 7. Control logic for generating control signal CR ( $UM_x$  = “unmapped” bit for register  $x$  and  $CM_x$  = “completed” bit for register  $x$ ).

[2], [6], these techniques entail additional complexities of tracking/predicting the last-use of registers and for maintaining and restoring the old state of the early-released registers if needed. To reduce the complexity of implementing recovery boosting, we put registers into the recovery boost mode only when they are in the *Unmapped* and *Mapped-Invalid* states. As we will show in Section VII, putting registers into the recovery boost mode just during these two states still provides substantial improvements in reliability.

Since the register renaming logic tracks whether a physical register is in the *Unmapped* or *Mapped-Invalid* state, the control signal for the recovery boost mode is set by the renaming logic. Each RAT entry has an ‘unmapped’ bit and a ‘completed’ bit that denotes whether the given physical register that the RAT entry points to has been mapped and whether it has been written to respectively. Using these two bits, we can implement the control signal for recovery boosting using the truth-table and the corresponding logic shown in Figs. 6 and 7 respectively.

#### B. Issue Queue

The issue queue houses instructions that have been fetched, decoded and renamed and are pending execution. Instructions are dynamically scheduled from the issue queue based on the availability of their source operands and functional units. Instruction issue consists of two steps—wakeup and select—which both need to be completed within a single clock cycle for high performance. Instructions that have finished execution broadcast their result tags to all the instructions in the issue queue. Each instruction in the issue queue compares the broadcasted tags with its own source tags for a possible match. Once both the source operand tags of an instruction have matched, the instruction is ready to be issued to a functional unit (instruction wakeup). A subset of the ready instructions are then selected to be issued to the functional units (instruction select). These instructions obtain their source input operands from the register file or the bypass network and then proceed to use the functional units granted to them.

We model a noncollapsing issue queue that is organized as a circular FIFO with head and tail pointers similar to the design proposed by Folegnani and Gonzalez [8]. We design the issue

queue entry to be similar to the one described by Palacharla *et al.* [13]. Conventional issue queues have a CAM/RAM structure where the CAM holds the source operand tags and the RAM holds the remaining information. The structure of the issue queue entry is shown in Fig. 8(a). Each entry has a valid-bit to indicate its status. The valid-bit is set when the entry is allocated for a dispatched instruction and is reset when the instruction is issued and leaves the issue queue. We put invalid entries into the recovery boost mode. (The valid-bit itself is not put into the recovery boost mode to ensure correct operation of the instruction scheduler). The CAM performs tag-matching operations against all the broadcasted tags each clock cycle. In order to do this, each CAM entry has a set of comparators and the number of comparators required depends on the issue width of the processor. The design of the CAM part of the issue queue for a single bit is shown in Fig. 8(b). In each cycle, each matchline is precharged. If there is a mismatch between the tag data in the memory cell and the broadcasted result tag in any of the CAM cells in the issue queue entry, then the corresponding matchline is discharged; otherwise the matchline stays high. If any of the matchlines for a given operand tag entry stays high, its corresponding ready signal (RDY) is asserted high via the OR-block shown in the figure.

To provide recovery boosting, the memory cells of the RAM and CAM structures are composed of the modified SRAM cells shown in Fig. 3(a). The modified issue queue entry is shown in Fig. 8(c). The valid-bit works as the control signal (CR) for the entry. When the memory cells in the entry transition to the valid state, the CR signal becomes high which pulls the ground down to low and the entry works in the normal operating mode. When the entry transitions to the invalid state, the CR signal becomes low and puts the memory cell into the recovery boost mode. When in the recovery boost mode, both nodes of the memory cells in both the RAM and the CAM parts are raised to  $V_{dd}$ . Due to the high node voltages, the comparators in the CAM will be triggered leading to a discharge of the matchline. To avoid this unnecessary precharging and discharging of the matchline (which wastes power), we further modify the issue queue entry so that the prechargers of the matchlines are connected to the CR signal. When CR is high in the normal operating mode, the matchlines will be precharged to  $V_{dd}$  and the tag-matching process will continue each cycle. When CR is low during the recovery boost mode, the matchlines stay low and therefore do not discharge.

### C. Circuit-Level Simulation Results

We now present the results from a circuit-level analysis of the designs discussed in Sections V-A and V-B. Please note that these simulation results are merely meant to substantiate our idea and are not meant to be an exhaustive analysis of the circuit and physical design of recovery boosting. We plan to conduct such in-depth studies in our future work.

We perform SPICE-level simulation using the Cadence Virtuoso Spectre circuit simulator to verify the functionality of our designs and determine their area and power consumption. Our experiments are carried out for the 32 nm process using the Predictive Technology Model. Our bitcell device

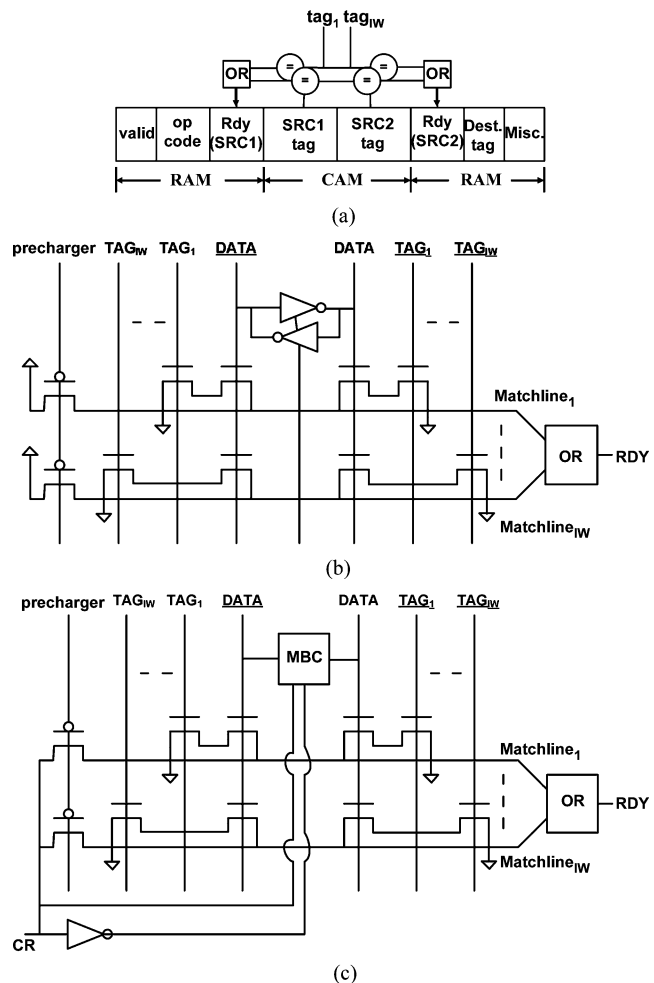


Fig. 8. Issue queue design. (a) An issue queue entry. (b) CAM structure of an issue queue entry (IW = issue width). (c) Modified CAM structure (IW = issue width). MBC is the modified bit-cell for recovery boosting.

sizes are: PMOS = 58 nm × 33, NMOS = 87 nm × 33, Access Transistor = 58 nm × 41. The supply voltage ( $V_{dd}$ ) and the operating temperature ( $T$ ) are 0.9 V and 90 °C, respectively. For each structure, we simulate two designs: the baseline design that uses conventional 6T SRAM cells (which do not provide recovery boosting) and the design that uses the modified SRAM cells discussed in Section IV to provide fine-grained recovery boosting.

1) *Functionality*: We evaluate two aspects of the functionality: 1) whether we can perform read, write and hold operations on the modified SRAM cells and 2) whether the modified cells correctly switch between the normal operating modes and the recovery boost mode. To evaluate these, we looked at the waveforms of the voltage variations at the nodes and bitlines of the cell in one clock cycle. We examine these waveforms for the read, write, and hold operations and also for the transitions between the recovery boost mode and the normal operating modes. We take into account the extra inverter delay required for changing the ground voltage of the cell during these transitions.

In Fig. 9, we show that the node voltages (Node0 and Node1) of the modified cell for recovery boosting takes 160 ps to reach desired voltage values whereas, the conventional 6T cell takes

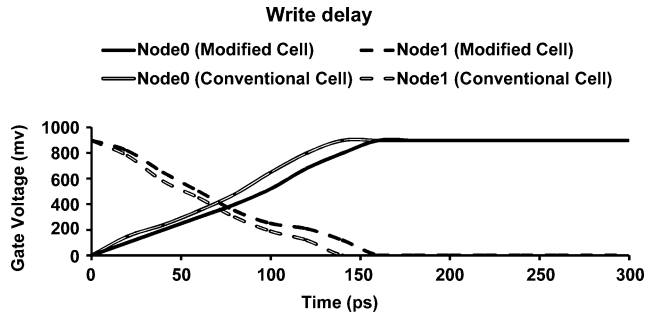


Fig. 9. Write delay of the modified bitcell. Node0 and Node1 are the node voltages of the bitcell ( $V_{dd} = 0.9$  V,  $T = 90$  °C).

140 ps. Even though the write delay is increased by 20 ps because of the increased capacitance in the modified cell, this operation can be done within a single cycle of a high performance processor. Since the read and hold operations behave in a similar way for both modified and conventional 6T cells, we do not present the waveforms for these operations.

Fig. 10(a) shows that the required time to switch to recovery boost mode is 190 ps. To switch to the recovery boost mode, both node voltages have to rise to  $V_{dd}$ . Node0 stays in  $V_{dd}$  and Node1 takes 190 ps to rise to  $V_{dd}$ . Since a bitcell transitions from recovery boost mode to normal operating mode on a write, we ran simulation to confirm that we can successfully write to the cell after the transition within a cycle. Fig. 10(b) shows the required time to switch to normal operating mode from recovery boost mode with a write operation. In recovery boost mode, both node voltages (Node0 and Node1) stays in  $V_{dd}$ . Therefore, with a write operation, Node0 has to stay in  $V_{dd}$  and Node1 has to be pulled down to GND. As we can see from the figure, Node1 reaches the desired value (GND) within 140 ps. Therefore, shifting to the recovery boost mode and to come back to the normal operating mode from it take 190 and 140 ps, respectively.

Our simulations indicate that we can correctly perform read, write, and hold operations on the registers and the issue queue entries that use the modified SRAM cells in the register file and the issue queue. The extra circuitry for recovery boosting is not active during the normal operating mode and therefore they do not interfere with normal operations on the cells. Since a register transitions from the recovery boost mode to the normal operating mode (i.e., *Mapped-Invalid* state to the *Mapped-Valid* state) on a write, our simulations confirm that we can successfully write to the cells after the transition. The same holds true for an issue queue entry. We also found that the matchlines for the CAM cells that are in the recovery boost mode are not precharged so that they never trigger a match for those cells.

2) *Clock Frequency Setting*: In our simulations, we found the smallest possible cycle-time for the modified SRAM cell to be 220 ps (a clock frequency of 4.5 GHz). We choose a more conservative cycle-time of 333 ps, which corresponds to a clock frequency of 3 GHz. We found the delay of the high- $V_t$  access transistors that connect to the  $V_{dd}$  rail to be small enough to transition the cell into the recovery boost mode within a single cycle for the 3-GHz clock frequency.

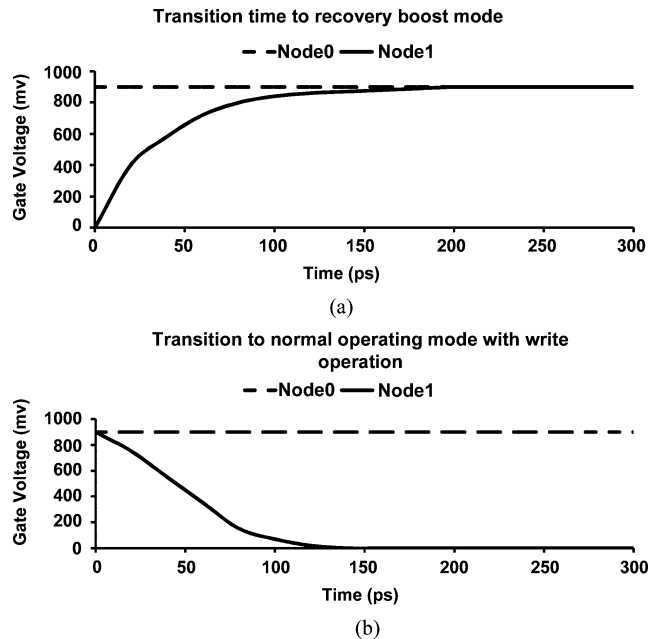


Fig. 10. Transition between recovery and normal modes. Node0 and Node1 are the Node voltages of the Bitcell ( $V_{dd} = 0.9$  V,  $T = 90$  °C). (a) Transition to recovery boost mode. (b) Transition to normal operating mode.

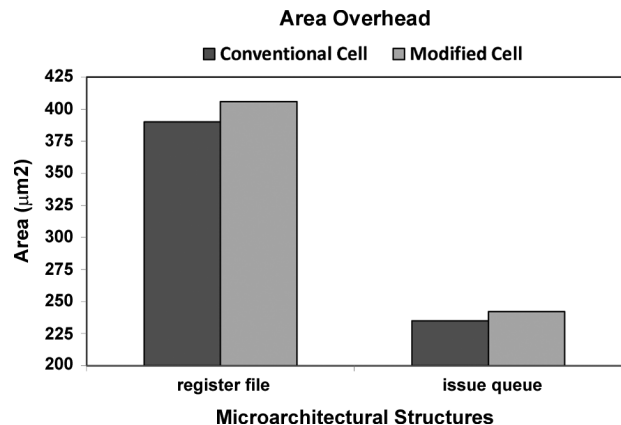


Fig. 11. Area of the register file and the issue queue for designs that use conventional 6T cells and cells modified to support recovery boosting.

3) *Area*: We designed our structures for both the baseline and recovery boosting cases to occupy the minimum area required to provide correct functionality. Care was taken to size the devices so that they are of minimal size while meeting the 3-GHz clock frequency requirement. We calculate the area of the structures based on the device sizes in their respective netlists. We assume that the area overhead due to any new routing or interconnect can be minimized by an optimized layout. In a typical SRAM array, the cells are laid out in the array in a mirrored fashion so that the same interconnect could be shared by adjacent rows and columns. Under these assumptions, the extra pMOS devices and the inverters would dominate the area overhead and accounting for the extra devices would give a first order approximation of the SRAM array area with the modified cells. The overheads for the multiported register file and issue queue are given in Fig. 11.

Since the register file has eight read-ports and four write-ports, each bitcell has 20 transistors: four transistors for the inverter-pair and eight transistors each for the write and read-ports (for supporting single-ended reads). Similarly, the issue queue has 4 read and write-ports respectively and has 16 transistors per bitcell. To support recovery boosting, we add two extra transistors of minimal size to each cell and one extra inverter for an entire row of 64 bitcells, in the case of the register file, or 65 bitcells for the issue queue. Therefore, adding the extra transistors for recovery boosting to these heavily multiported structures is expected to add only a small amount of area. Indeed, we can see that the area of the physical register file and issue queue that use the modified cells are 4% and 3% respectively more than their baseline designs. This overhead is roughly equivalent to the area occupied by three registers in the modified register file and two entries in the modified issue queue. We can therefore design the register file and issue queue to be area-neutral with respect to the baseline (i.e., occupy the same area as the baseline design) by having their capacities reduced by three registers and two entries respectively. The rationale behind going in for area-neutral structures is to minimize the impact of designing structures that employ recovery boosting on the processor floorplan. Going for the area-neutral design of the structures could affect the performance of the processor. The performance impact of these area-neutral designs are evaluated in Section VII.

4) *Dynamic and Leakage Power Consumption*: Fig. 12 gives the power consumption of a single register and a single issue queue entry for both the baseline design and the one that uses the modified SRAM cell. For the register, we show the power consumed for the read, write and hold operations as well as when the cells are in the recovery boost mode. For the issue queue entry, in addition to the power consumed in the recovery boost mode, we quantify the power consumed in each of the three normal operating modes. For each of these modes, we present the power consumption for two scenarios: 1) when both source tags of an entry mismatch with the ones broadcast down the issue queue in the same cycle, which is the highest power consumption scenario since all the matchlines discharge and 2) when both source tags match in the same cycle, which consumes the least amount of power.

We can see that the power consumed by the designs that use the modified SRAM cells for the read, write and hold operations are nearly equal to those of the baseline designs. The maximum increase in power is less than 1% for the register and less than 2% for the issue queue entry. The power consumption of the issue queue entry is higher than the register because of its CAM/RAM structure. The slight increase in power for the recovery boost designs is due to leakage in one of the pMOS access transistors that connect to the  $V_{dd}$  rail. The sources of the pMOS access transistors are connected to  $V_{dd}$  and the drains are connected to the nodes. Therefore, based on whether a cell holds a “0” or a “1,” one of the two pMOS devices will leak. Since we use high- $V_t$  pMOS devices as the access transistors for the cells (to reduce the impact of NBTI), the leakage power of these transistors is also reduced.

In memory arrays that use conventional SRAM cells, the cells will normally be operating in the hold mode when they house invalid data. However, when the modified cells are used, cells that

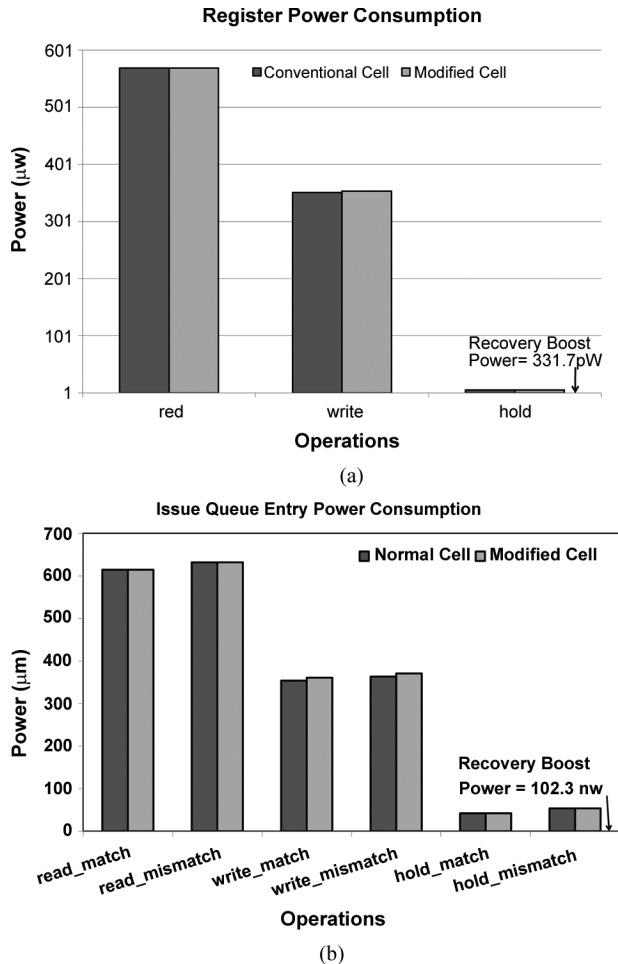


Fig. 12. Power consumption of a single register and a single issue queue entry ( $V_{dd} = 0.9$  V,  $T = 90$  °C). (a) Register. (b) Issue queue entry.

hold invalid data can operate in the recovery boost mode. We can see that the power consumed in the recovery boost mode is orders of magnitude less than in the hold mode. This is because the recovery boost operation raises Node0 and Node1 [shown in Fig. 3(a)] and the ground to  $V_{dd}$ , which cuts off the path from  $V_{dd}$  to ground and significantly reduces the leakage currents. Finally, there is a small power benefit at the structure level since we use area-neutral designs for the physical register file and the issue queue that are slightly smaller than the baseline designs.

## VI. EXPERIMENTAL METHODOLOGY FOR THE ARCHITECTURE-LEVEL ANALYSIS

Having seen the circuit-level design of the physical register file and the issue queue to support recovery boosting, we now evaluate the impact of using these techniques at the architecture level. We carry out our architecture-level evaluations via execution-driven simulation using the M5 simulator [3]. We use the system-call emulation mode of M5. Our workloads consist of all 26 benchmarks from the SPEC CPU2000 benchmark suite. The benchmarks are compiled for the Alpha ISA and use the reference input set. We perform detailed simulation of the first 100-million instruction SimPoint for each benchmark [18]. We model a four-wide issue core, which is similar to those in multicore processors. We assume the initial threshold voltage of the

pMOS devices in the memory cells to be 0.2 V and the service life of the processor to be seven years based on the work by Tiwari and Torrellas [22].

1) *Reliability Metric—Read Static Noise Margin (SNM)*: NBTI causes an increase in the threshold voltage of the pMOS transistors. In the case of SRAM cells, this shift in  $V_t$  could increase the time needed for reading from and writing to the cells. NBTI could also decrease the read SNM of the cells. The SNM is a measure of the stability of the cell and specifies the maximum amount of voltage noise that can be tolerated at the nodes of the memory cell before the contents of the cell get flipped [16]. Previous work [11] has shown that, of these three metrics, the SNM is the one that is most heavily affected by NBTI and therefore we use SNM as the reliability metric in this paper.

Initially, before the processor is used for executing workloads, the bitcells in the register file and the issue queue are designed such that their SNM is not limited by the strength of the pMOS devices. But after these structures are exercised by workloads, their SNM gets limited by the strength of the pMOS devices due the impact of NBTI on  $V_t$ . We capture this impact by tracking the stress and recovery cycles on all of the pMOS devices in the register file and the issue queue (based on our circuit-level designs of these structures) over the course of an architecture simulation and extrapolate the statistics to calculate the degradation in  $V_t$  after the seven-year service life. We then feed the  $V_t$  values of these pMOS devices into the Cadence Virtuoso Spectre circuit simulator to calculate the SNM of all the cells in a structure at the end of the seven-year period and use the smallest value to denote the SNM for that structure.

## VII. ARCHITECTURE-LEVEL SIMULATION RESULTS

We now study the impact of putting memory cells of registers and issue queue entries into the recovery boost mode when they hold invalid data. As discussed earlier, we put registers into the recovery boost mode when they are in the *Unmapped* and *Mapped-Invalid* states. We present results only for the integer register file for two reasons. First, the integer benchmarks have very few floating-point instructions and therefore rarely exercise the floating-point register file. Second, several of the floating-point benchmarks have a large number of integer instructions in their first 100-million instruction SimPoint and therefore significantly exercise the integer register file. (Nine out of the 14 floating-point workloads have more integer than floating-point instructions in their first SimPoint.) As a result, the integer register file is exercised to a greater extent than the the floating-point register file for most of the benchmarks. Issue queue entries are put into the recovery boost mode when they no longer hold valid data.

We evaluate three different processor configurations, which we denote as: *Baseline*, *Recovery Boosting* and *Balancing*. *Baseline* models four-wide issue core that do not use any NBTI mitigation technique. *Recovery Boosting* replaces the integer register file and the issue queue of the baseline configuration with their counterparts that support recovery boosting. We assume that the modified structures for *Recovery Boosting* are designed to be area-neutral with respect to *Baseline* by trading off a small

amount of storage capacity to accommodate the extra area required to implement recovery boosting. Based on our area evaluations in Section V, we assume that the modified integer register file and issue queue have 125 registers and 62 entries, respectively. In all of our simulations, we find that this reduction in capacity has a negligible impact on performance and therefore we do not present detailed performance results. *Balancing* denotes a recovery enhancement scheme similar to the one proposed in [1] that uses the same time intervals that *Recovery Boosting* exploits to balance the degradation of the two pMOS devices in the memory cell. As pointed out by Abella *et al.* [1], flipping the contents of the memory cells only when they hold invalid data instead of when they hold both valid and invalid data, for which additional circuitry is required [11], is the preferable approach for high-speed SRAM structures in order to not increase their delay significantly. Since the access times of the physical register file and the issue queue have a strong impact on processor performance, the *Balancing* technique is applied only when the memory cells hold invalid data. We optimistically assume that *Balancing* does not impose any additional area overheads over the baseline design and that it can keep the inputs to each pMOS device at a logic “0” exactly 50% of the time whenever the cells are in this mode.

For the remainder of this paper, we will refer to the integer register file and the issue queue as RF and IQ, respectively.

### A. Physical Register File Results

Fig. 13 shows the  $V_t$  degradation and the resultant SNM for the RF across the benchmarks due to NBTI after the seven-year service life for the *Baseline*, *Recovery Boosting*, and *Balancing* configurations. As mentioned earlier, the initial  $V_t$  is 200 mV and, using this value, we get the initial SNM of the RF to be 171 mV before the RF is stressed by the benchmark. For the *Baseline*, on an average, the  $V_t$  degrades to 305 mV across the benchmarks, which leads to an average SNM of 109 mV. Therefore, *Baseline* causes about 37% degradation in SNM. To improve the SNM over the *Baseline*, *Recovery Boosting* takes advantage of the invalid periods of the RF to apply recovery to the registers. A breakdown of the time spent by the registers in the four different states given in Fig. 5 is shown in Fig. 14. The values given in the graph are an average over all the registers and over the entire SimPoint for each benchmark. As we can see, the registers are in the *Unmapped* and *Mapped-Invalid* states for a large fraction of time for most of the benchmarks. Therefore, we have significant opportunities to apply recovery boosting for the RF. The impact on the SNM as a result of using recovery boosting is given in Fig. 13. As Fig. 13 shows, the average degraded  $V_t$  stays close to 245 mV because of the applied recovery to the RF bitcells. This causes the SNM to degrade to an average value of 144 mV, which is about 15% degradation in SNM over the initial condition. Similarly, for the *Balancing* approach, the average degraded  $V_t$  stays close to 265 mV and the SNM degrades to an average value of 133 mV.

In Fig. 15, each pair of bars shows the improvement in the SNM over *Baseline* for *Recovery Boosting* and *Balancing*, respectively. As Fig. 15 shows, while *Balancing* provides a good improvement in the SNM, *Recovery Boosting* provides significantly higher reliability benefits by virtue of its ability to

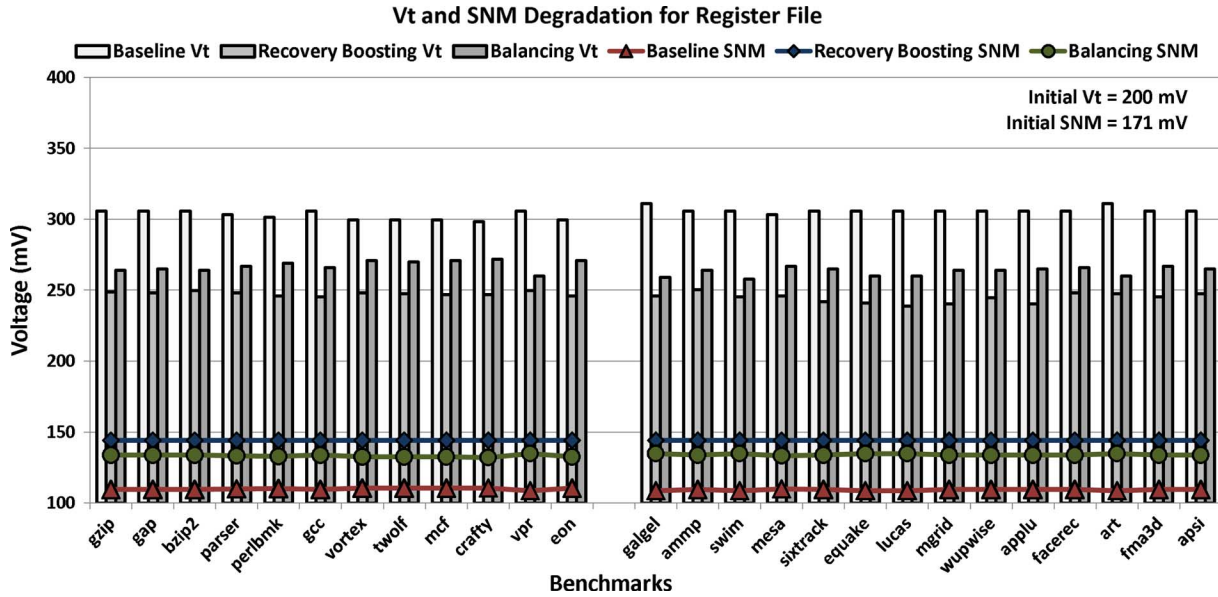


Fig. 13.  $V_t$  and SNM degradation for the RF for the *Baseline*, *Recovery Boosting*, and *Balancing* configurations ( $V_{dd} = 0.9$  V,  $T = 90$  °C).

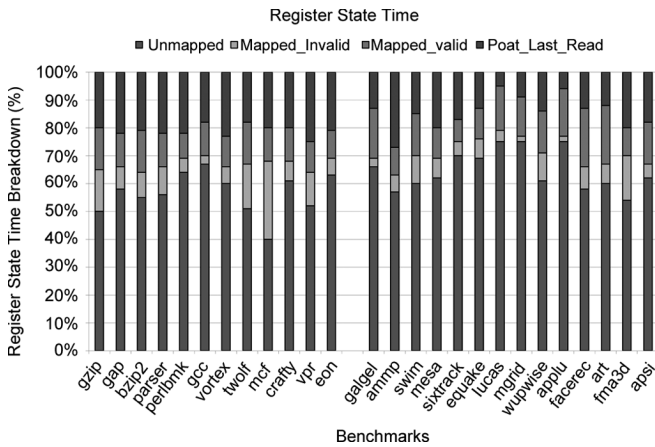


Fig. 14. Breakdown of time spent by the registers in different states. The lowest part of each stacked bar is the *Unmapped* state.

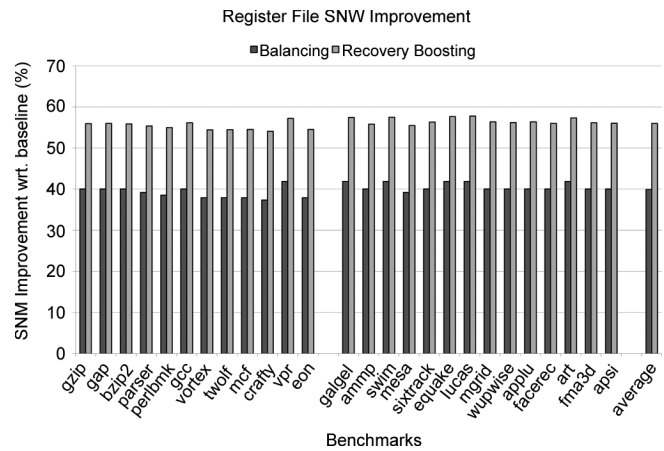


Fig. 15. Improvement in the SNM for the RF over the *Baseline* processor configuration ( $V_{dd} = 0.9$  V,  $T = 90$  °C).

put both pMOS devices into the recovery mode. Across all the benchmarks, *Balancing* provides a 40% improvement in the SNM while *Recovery Boosting* provides a 56% improvement. These results clearly highlight the benefits of recovery boosting as a technique to mitigate NBTI in the RF.

### B. Issue Queue Results

Fig. 16 shows the  $V_t$  degradation and the resultant SNM for the IQ across the benchmarks due to NBTI for the *Baseline*, *Recovery Boosting*, and *Balancing* configurations. Similar to the RF, the initial  $V_t$  and the SNM are 200 and 171 mV, respectively. On an average, the  $V_t$  degrades to 305 mV and the SNM degrades to 109 mV for the *Baseline*. Unlike the RF, the  $V_t$  and the SNM of the IQ varies across different benchmarks for the *Recovery Boosting* configuration. Specially, the  $V_t$  and the SNM varies a lot for the floating-point benchmarks. The reason behind this is illustrated in Fig. 17 where the breakdown of the time spent by the IQ entries in the *Valid* and *Invalid* states for each benchmark is shown. The time-breakdown is an average over

all the entries in the IQ and over the entire SimPoint of each benchmark. The duration of the *Invalid* state is much shorter for some floating-point benchmarks as shown in the figure. On an average,  $V_t$  degrades to 254 mV and the resultant average SNM becomes 140 mV for the *Recovery Boosting* configuration. Similarly,  $V_t$  degrades to 272 mV and the resultant average SNM becomes 131 mV for the *Balancing* configuration.

While the RF results show that *Recovery Boosting* is consistently superior to *Balancing*, the IQ shows a more varied trend. The reliability results are given in Fig. 18. For the integer benchmarks, which are in the left-hand side group in the graphs in Fig. 17, the IQ entries are in the *Invalid* state for a large fraction of the time (over 76% on average) and therefore enjoy significant benefits from recovery boosting. The exception to this is *mcf*, where the IQ entries spend only 34% of the time in the *Invalid* state. The difference in the SNM improvement between *Balancing* and *Recovery Boosting* is much smaller for this benchmark and so is the overall benefit over *Baseline*. This is because 37% of the instructions in *mcf* are memory instructions

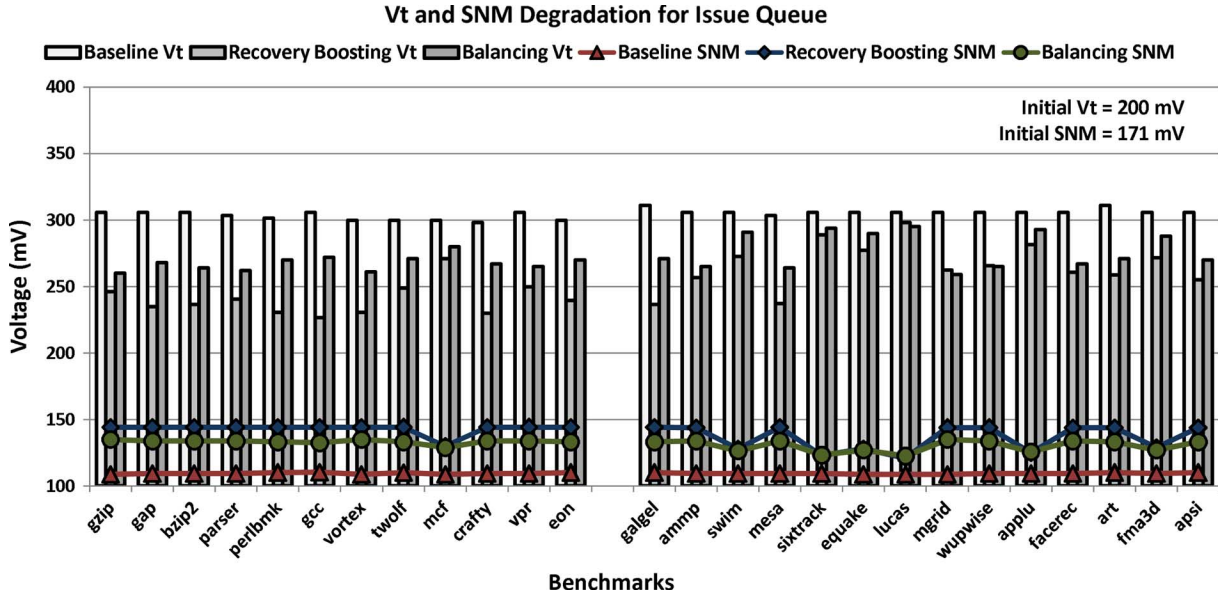


Fig. 16.  $V_t$  and SNM Degradation for the IQ for the *Baseline*, *Recovery Boosting* and *Balancing* configurations ( $V_{dd} = 0.9$  V,  $T = 90$  °C).

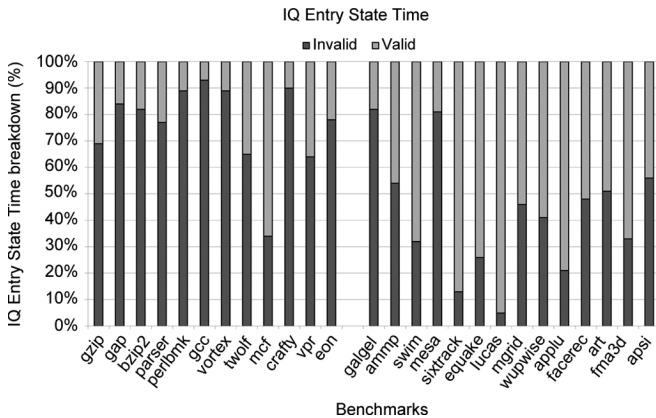


Fig. 17. Breakdown of time spent by the IQ entries in the *Valid* and *Invalid* states.

(33.5% are loads and 3.5% are stores). Given the large number of load-instructions and the fact that loads values are consumed by subsequent instructions in its dependence-chain that wait in the IQ, they have a significant impact on the occupancy characteristics of the IQ. We find that the L1 data-cache miss rate for *mcf* is high (54.5%), which leads to these dependent instructions occupying IQ entries for a longer period of time. As a consequence, there are fewer opportunities to apply either *Balancing* or *Recovery Boosting* to the IQ for *mcf* compared to the other integer benchmarks. Note that although the IQ entries are in the *Valid* state while these instructions wait, the destination register of a producer instruction remains in the *Mapped-Invalid* state in the RF till the value is actually ready to be written to the register. We can see in Fig. 14 that the registers spend a longer time in the *Mapped-Invalid* state for *mcf* than for the other integer benchmarks. Therefore, the RF still benefits from recovery boosting during this period.

The floating-point workloads exhibit a range of behaviors. *Recovery Boosting* provides significant improvements in the SNM for more than half the benchmarks and the benefits

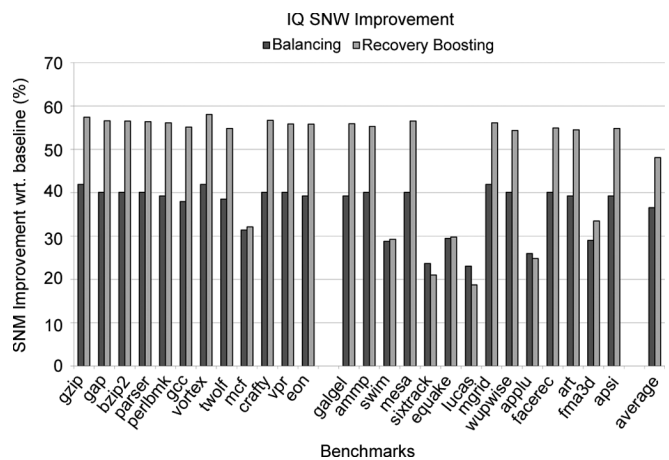


Fig. 18. Improvement in the SNM for the IQ over the baseline processor configuration ( $V_{dd} = 0.9$  V,  $T = 90$  °C).

provided are much better than *Balancing*. However, for certain workloads (*swim*, *quake*, *fma3d*) the benefits provided by *Recovery Boosting* and *Balancing* are nearly the same. Most interestingly, for *sixtrack*, *lucas* and *applu*, *Balancing* provides slightly *better* improvement in the SNM than *Recovery Boosting*. We now explain why this happens.

First, as we expect, there is a relationship between the benefits that *Recovery Boosting* provides and the amount of time that the IQ entries spend in the *Invalid* state, as Fig. 17 indicates. There are several factors that influence the amount of time that various workloads spend in the *Invalid* state, such as the percentage of high-latency instructions in the instruction mix of the benchmark and the memory system performance (both the hit-rates of the L1 instruction and data caches as well as the average miss latencies). For workloads that have a significant number of high-latency floating-point instructions, the instructions data-dependent on those high-latency instructions will occupy IQ entries for a longer time. *sixtrack*, *lucas*

and `applu` contain a significant number of floating-point instructions in their instruction-mix (65%, 64%, and 52% respectively). These three workloads also have the highest fraction of floating-point multiply instructions among all the floating-point workloads (36%, 23%, and 24% of the overall mix, respectively). Since a floating-point multiply takes four cycles to execute, instructions on their dependence-chains wait in the IQ for a long time in the *Valid* state. Similarly, for workloads that have a high percentage of load instructions, the IQ entries occupied by the loads and their dependence-chains will remain in the *Valid* state for a longer time if the loads frequently miss in the cache. In general, the IQ occupancy behavior depends on both the characteristics of the application and the exact microarchitectural configuration of the processor. However, since we do not optimize either of these factors for *Balancing* and *Recovery Boosting*, it is more important to understand when one scheme will be more beneficial than the other in terms of reliability.

To achieve a good SNM, it is important to minimize the increase in  $V_t$  due to NBTI of both pMOS devices in the memory cell. Additionally, it is also important to ensure that the *difference* in the threshold voltages between the two pMOS devices is kept as small as possible. *Recovery Boosting* addresses the first condition whereas *Balancing* addresses the second. When an IQ entry holds valid data for a long period of time, as is the case for `sixtrack`, `lucas` and `applu`, one of the pMOS devices in each memory cell stays in the stress phase while the other in the recovery phase. This increases the difference in the  $V_t$  between the two pMOS devices and therefore degrades the SNM. When the IQ entry is in the *Invalid* state for a short period of time, *Recovery Boosting* reduces the  $V_t$  of both pMOS devices all the memory cells. However, *Balancing* flips the bits stored in those cells so that the pMOS device that was in the recovery phase while the IQ entry was valid now enters the stress phase and vice-versa, which tends to reduce the difference in  $V_t$  between the two pMOS devices. As a result, for workloads where the IQ entries spend very little time in the *Invalid* state, the SNM for the *Balancing* scheme is better than for *Recovery Boosting*. The difference is especially evident for `lucas`, whose IQ entries spent the least amount of time in the *Invalid* state (5% of the time) as shown in Fig. 17. However, when the *Invalid* periods are even slightly longer, the difference between the two schemes becomes less evident and it starts becoming more beneficial to reduce  $V_t$  than to try maintaining a small threshold voltage difference for longer *Invalid* periods. For example, we can see in Fig. 17 that `sixtrack` and `applu` spend successively longer periods in the *Invalid* state (13% and 21% respectively) and consequently the gap between *Balancing* and *Recovery Boosting* diminishes and *Recovery Boosting* provides more benefit than *Balancing* for workloads with longer *Invalid* times.

In general, we find that *Recovery Boosting* is more beneficial than *Balancing* for a wide range of IQ entry *Invalid*-state occupancy times. *Recovery Boosting* provides a 48% improvement in the SNM over *Baseline* whereas *Balancing* provides only a 37% improvement for the entire SPEC CPU2000 benchmark suite.

### VIII. CONCLUSION AND FUTURE WORK

NBTI is one of the most important silicon reliability problems facing processor designers. SRAM memory cells are especially

vulnerable to NBTI since the input to one of the pMOS devices in the cell is always at a logic “0.” In this paper, we propose recovery boosting, a technique that allows both pMOS devices in the cell to be put into the recovery mode by raising the ground voltage and the bitline to  $V_{dd}$ . We show how fine-grained recovery boosting can be used to design the physical register file and issue queue and evaluate their designs via SPICE-level simulations. We then show that area-neutral designs of these two structures can provide significant reliability benefits with very little impact on power consumption and negligible loss in performance.

The fine-grained recovery boosting approach that we evaluate in this paper can be used for small SRAM arrays. In future work, we plan to study the use of coarse-grained recovery boosting, which imposes less area overheads, for designing caches. Caches pose additional challenges, such as identifying when lines become valid to put them into the recovery boost mode. We plan to explore the use of techniques such as dead-block prediction [10] in conjunction with recovery boosting to mitigate the impact of NBTI on caches. We also plan to study the circuit-level design of recovery boosting in depth.

### ACKNOWLEDGMENT

The authors would like to thank M. Stan, A. Cabe, B. Calhoun, and R. Mann for their valuable inputs.

### REFERENCES

- [1] J. Abella, X. Vera, and A. Gonzalez, “Penelope: The NBTI-aware processor,” in *Proc. 40th IEEE/ACM Int. Symp. Microarchitecture*, 2007.
- [2] H. Akkary, R. Rajwar, and S. T. Srinivasan, “Checkpoint processing and recovery: Towards scalable large instruction window processors,” in *Proc. Int. Symp. Microarchitecture (MICRO)*, Dec. 2003, pp. 423–434.
- [3] N. L. Binkert et al., “The M5 simulator: Modeling networked systems,” *IEEE Micro*, vol. 26, no. 4, pp. 52–60, Jul. 2006.
- [4] P. Bose, J. Shin, and V. Zyuban, “Method for Extending Lifetime Reliability of Digital Logic Devices Through Removal of Aging Mechanisms,” U.S. Patent 7 489 161, Feb. 10, 2009.
- [5] A. Cabe, Z. Qi, S. Wooters, T. Blalock, and M. Stan, “Small embeddable NBTI sensors (SENS) for tracking on-chip performance decay,” in *Proc. Int. Symp. Quality Electron. Design (ISQED)*, Mar. 2009, pp. 1–6.
- [6] O. Ergin, D. Balkan, D. Ponomarev, and K. Ghose, “Increasing processor performance through early register release,” in *Proc. Int. Conf. Comput. Design (ICCD)*, Oct. 2004, pp. 480–487.
- [7] S. Feng, S. Gupta, and S. Mahlke, “Olay: Combat the signs of aging with introspective reliability management,” in *Proc. Workshop Quality-Aware Design (W-QUAD)*, 2008.
- [8] D. Folegnani and A. Gonzalez, “Energy-effective issue logic,” in *Proc. Int. Symp. Comput. Architecture (ISCA)*, Jun. 2001, pp. 230–239.
- [9] X. Fu, T. Li, and J. Fortes, “NBTI tolerant microarchitecture design in the presence of process variation,” in *Proc. Int. Symp. Microarchitecture (MICRO)*, Nov. 2008, pp. 399–410.
- [10] S. Kaxiras, Z. Hu, and M. Martonosi, “Cache decay: Exploiting generational behavior to reduce cache leakage power,” in *Proc. Int. Symp. Comput. Architecture (ISCA)*, Jun. 2001, pp. 240–251.
- [11] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, “Impact of NBTI on SRAM read stability and design for reliability,” in *Proc. Int. Symp. Quality Electron. Design*, 2006, pp. 210–218.
- [12] Y. Li, D. Brooks, Z. hu, and K. Skadron, “Performance, energy and thermal considerations for SMT and CMP architectures,” in *Proc. Int. Symp. High-Performance Comput. Architecture (HPCA)*, 2005, pp. 71–82.
- [13] S. Palacharla, “Complexity-effective superscalar processors,” Ph.D. dissertation, Dept. Comput. Sci., Univ. of Wisconsin, Madison, 1998.

- [14] S. Park, K. Kang, and K. Roy, "Reliability implications of bias-temperature instability in digital ICs," *IEEE Design Test of Comput.*, pp. 8–17, Dec. 2009.
- [15] G. Reimbold *et al.*, "Initial and PBTI-induced traps and charges in Hf-based oxides/TiN stacks," *Microelectron. Reliabil.*, vol. 47, no. 4–5, pp. 489–496, Apr. 2007.
- [16] E. Seevinck, F. J. List, and J. Lohstroh, "Static-noise margin analysis of MOS SRAM cells," *IEEE J. Solid-State Circuits*, vol. 22, no. 5, pp. 748–754, Oct. 1987.
- [17] J. P. Shen and M. H. Lipasti, *Modern Processor Design: Fundamentals of Superscalar Processors (Beta Edition)*. New York: McGraw Hill, 2003.
- [18] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proc. Int. Conf. Architect. Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 2002, pp. 45–57.
- [19] J. Shin, V. Zyuban, P. Bose, and T. M. Pinkston, "A proactive wearout recovery approach for exploiting microarchitectural redundancy to extend cache SRAM lifetime," in *Proc. Int. Symp. Comput. Architecture*, 2008, pp. 353–362.
- [20] A. Sil, S. Ghosh, N. Gogineni, and M. Bayoumi, "A novel high write speed, low power, read-SNM-free 6T SRAM cell," in *Proc. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2008, pp. 771–774.
- [21] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," in *Proc. Int. Symp. Comput. Architecture (ISCA)*, Jun. 2004, pp. 276–287.
- [22] A. Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," in *Proc. Int. Symp. Microarchitecture (MICRO)*, Nov. 2008, pp. 129–140.
- [23] W. Wang, V. Reddy, A. T. Krishnan, R. Vattikonda, S. Krishnan, and Y. Cao, "Compact modeling and simulation of circuit reliability for 65-nm CMOS technology," *IEEE Trans. Device Mater. Reliabil.*, vol. 7, no. 4, pp. 509–517, 2007.
- [24] X. Yang, E. Weglarz, and K. Saluja, "On NBTI degradation process in digital logic circuits," in *Proc. Int. Conf. VLSI Design*, Jan. 2007, pp. 723–730.

**Taniya Siddiqua** received the B.Sc. degree in computer science and engineering from Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2005, and the M.s. degree in computer science from the University of Virginia, Charlottesville, in 2009, where she is currently working toward the Ph.D. degree in computer science.

She served as a Lecturer with BRAC University, Dhaka, Bangladesh, from 2006 to 2007. Her research interests include processor fault tolerance and VLSI design.

**Sudhanva Gurumurthi (SM'10)** received the B.E. degree from the College of Engineering Guindy, Chennai, India, in 2000, and the Ph.D. degree from Pennsylvania State University, University Park, in 2005, both in computer science and engineering.

He has held research positions with IBM Research and Intel Corporation and has served as a faculty consultant for Intel. His research interests include storage and memory systems and processor fault tolerance. He has served on the program and organizing committees of several top computer architecture and systems conference, including ISCA, ASPLOS, HPCA, FAST, and SIGMETRICS.

Dr. Gurumurthi is a member of the Association for Computing Machinery. He was a recipient of the National Science Foundation (NSF) CAREER Award and several research awards from NSF, Google, Intel, and Hewlett-Packard. He is an associate editor-in-chief of IEEE COMPUTER ARCHITECTURE LETTERS.