DESIGN OF NEGATIVE BIAS TEMPERATURE INSTABILITY (NBTI) TOLERANT REGISTER FILE

by

Saurabh Kothawade

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Engineering

Approved:

Dr. Koushik Chakraborty Major Professor	Dr. Sanghamitra Roy Committee Member
Dr. Edmund Spencer Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2011

Abstract

Design of Negative Bias Temperature Instability (NBTI) Tolerant Register File

by

Saurabh Kothawade, Master of Science

Utah State University, 2011

Major Professor: Dr. Koushik Chakraborty
Department: Electrical and Computer Engineering

Degradation of transistor parameter values due to Negative Bias Temperature Instability (NBTI) has emerged as a major reliability problem in current and future technology generations. NBTI Aging of a Static Random Access Memory (SRAM) cell leads to a lower noise margin, thereby increasing the failure rate. The register file, which consists of an array of SRAM cells, can suffer from data loss, leading to a system failure. In this work, we study the source of NBTI stress in an architecture and physical register file. Based on our study, we modified the register file structure to reduce the NBTI degradation and improve the overall system reliability. Having evaluated new register file structures, we find that our techniques substantially improve reliability of the register files. The new register files have small overhead, while in some cases they provide saving in area and power.

(58 pages)

Public Abstract

Design of Negative Bias Temperature Instability (NBTI) Tolerant Register File

by

Saurabh Kothawade, Master of Science

Utah State University, 2011

Major Professor: Dr. Koushik Chakraborty
Department: Electrical and Computer Engineering

Negative Bias Temperature Instability (NBTI) is becoming a major reliability problem in the semiconductor industry. As time passes, NBTI reduces the capacity of performing correct computations in the microprocessor. Hence, after certain time period, the microprocessor may fail to work as we expect, causing failure of the entire system it is part of. In this research, we study the root cause of the failure due to NBTI effect. Based on our findings, we propose multiple methods to reduce the negative impact of NBTI on a microprocessor. We build a comprehensive experimental setup to consider real world effects in a microprocessor. We evaluate our methods against the previous work and find that our methods substantially improve the processor reliability. This research could be useful in the future to extend lifetime of the processor.

To my parents...

Acknowledgments

I express my sincere gratitude towards my major professor, Dr. Koushik Chakraborty, for providing me an opportunity to work on exciting topics. His support was critical in my thesis work. This thesis would have been impossible without his knowledge, guidance, and feedback. I would also like to thank other committee members, Dr. Roy and Dr. Spencer.

I thank other members of the VLSI research group, Kshitij, Satyajit, Yiding, Dean, and Jason for their help. It was fun working with them.

I express deep appreciation for my family back in India, my parents and my elder brother, for giving full support and having faith in me. I thank my brother for always being a constant source of inspiration.

Saurabh Kothawade

Contents

List of Tables

List of Figures

Acronyms

NBTI	Negative Bias Temperature Instability
BTI	Bias Temperature Instability
PV	Process Variation
SRAM	Static Random Access Memory
DRV	Data Retention Voltage
BL	Bit Line
WL	Write Line
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
PMOS	p-channel MOSFET
NMOS	n-channel MOSFET
SNM	Static Noise Margin
ZBP	Zero Bias Probability
OBP	One Bias Probability
INV	Bit Flip Technique
OS	Operating System
RR	Register Rotation
BR	Bit Rotation
RBR	Register Bit Rotation
TSMC	Taiwan Semiconductor Manufacturing Company
ZP	Zero Predominance
NP Predictor	Non-Zero Predominance Predictor
ISA	Instruction Set Architecture
SPEC	Standard Performance Evaluation Corporation

Chapter 1
Introduction

Negative Bias Temperature Instability (NBTI) has emerged as a major reliability challenge for the semiconductor industry in recent years. NBTI impact is getting worse in each technology generation with greater performance and reliability loss. When a negative voltage is applied at a p-channel transistor (PMOS) gate, interface traps are formed near oxide layer, causing a change in transistor characteristics. When the input to a PMOS is low (logic zero), the transistor is in a *stress* phase. During the stress phase, the transistor parameters slowly deviate from the nominal value. When the input to the PMOS is high (logic one), the transistor is in a *recovery* phase. During the recovery phase, trapped charges are released, regaining the original transistor state. The PMOS enters into stress and recovery phases alternately, when the input to the PMOS is dynamic. Longer the stress period, higher is the impact of NBTI on transistor parameters. Therefore, input to the transistor indirectly determines the extent of NBTI degradation.

Static Random Access Memory (SRAM) cells, which are the key elements in register files and caches, are severely affected due to the NBTI aging. An SRAM cell storing the same value for a large period of time undergoes highly unbalanced stress, causing a substantial reduction in its reliability characteristics. This degradation of reliability of the SRAM cell can result in a loss of the stored value. Therefore, register files and caches are highly prone to failures due to NBTI. Recent works have proposed techniques to improve reliability in the physical register file by extending the recovery period during idle cycles [1, 2]. They manipulate bit cell contents during the idle periods of physical registers to relax one or more PMOS transistors. However, the effectiveness of these techniques strongly depend on the length of the available idle period. A power-efficient physical register file, where its total capacity is closer to the architectural register file size, is likely to have a substantially shorter idle period, thereby limiting the reliability boost achievable through these techniques.

In our work, we target NBTI degradation in both architecture and physical register files. To improve the reliability of the physical register file, we use an orthogonal approach with instruction level analysis, instead of exploiting idle cycles. We investigate NBTI stress from the output values generated during instruction execution and its propagation in the register file. Based on this approach, we observe a wide variability in the NBTI stress induced by different instructions in a physical register file. While some instructions inherently produce high stress output values, others generate substantially lower NBTI stress. Using this approach, we design predictors to detect instructions producing large NBTI stress, and design an NBTI tolerant physical register file.

The nature of NBTI degradation in an architecture register file is different from that in a physical register file. To mitigate the NBTI degradation in an architecture register file, we study the stress pattern generated by interleaving of applications. The resultant degradation is worse than perceived by analyzing applications in isolation. Recently proposed techniques of periodic inversion of stored bits [1, 3] are unable to mitigate such realistic use scenarios. While these techniques tackle the bias in the bit pattern, they perform poorly when the bit patterns exhibit wide variability.

We analyze the differential stress in an architecture register file using an end-to-end approach through a comprehensive circuit-architectural analysis of SRAM cells. We analyze the behavior pattern by running a sequence of applications on a microprocessor, mimicking the real life scenario of a typical desktop computer system. Our end-to-end approach, which is able to simultaneously model multiple layers of system design abstractions (applications, operating systems, architecture, circuit), provides a more realistic modeling of NBTI degradation in a register file.

To mitigate the limitations of the existing techniques for NBTI mitigation in a register file, we propose several micro-architecture techniques to uniformly spread out both the *inherent bias* and *variability* of the bit patterns across the entire pool of SRAM cells in the register file. Our techniques are able to achieve substantially robust aging characteristics, across a wide spectrum of use scenarios.

Key contributions of this work are as follows:

- First, we study the NBTI effect and its impact on an SRAM cell. We measure reliability of the cell in terms of Static Noise Margin (SNM). We identify the input probability as a primary factor of the reliability degradation and study its relation with SNM. Further, we evaluate the benefits of the transistor sizing and supply voltage scaling for NBTI mitigation.

- We investigate the nature of the NBTI degradation in an architecture and physical register file. We find source of NBTI stress and propose techniques to minimize the NBTI stress in each register file.

- To measure impact of the NBTI at the processor level, we build a circuit-architecture simulation framework. This framework combines transistor simulations, Register Transfer Level (RTL) synthesis, and full-system architecture simulations to give system level idea of the NBTI degradation for real programs on a typical desktop environment.

- We evaluate performance of our proposed techniques for various configurations and compare them with the previous work. We find that our techniques provide better robustness at lower overhead for a variety of register file design. Our techniques for the architecture register file reduce the reliability degradation by 2.2X and lower the uncertainity by 14X. Similarly, the modifications in the physical register file improve the reliability by 125% at 22nm technology node.

The remainder of this thesis is organized as follows: Chapter 2 covers the previous work discussing the NBTI effect and its impact on Very Large Scale Integrated (VLSI) circuits. In Chapter 3, we discuss NBTI impact on the reliability of an SRAM cell. In Chapter 4, we study NBTI impact in an architecture register file. This chapter includes description of our proposed techniques and results showing improvement in the reliability. In Chapter 5, we analyze the NBTI stress in a physical register file. We propose modifications in the physical register file structure and register renaming policy. This chapter shows results of SNM improvement in the physical register file due to our techniques. We complete this report with a conclusion in Chapter 6.

Chapter 2

Related Work

Several papers have discussed NBTI mechanism and its impact on transistor parameters [4–7]. Bhardwaj et al. proposed a predictive model to calculate shift in threshold voltage of transistor due to aging [8]. Recent work has proposed numerical simulation engine for NBTI-induced aging, based on reaction-diffusion model [9].

Memory circuits are severely affected by aging as its transistors are less frequently switched than logic circuits. Reddy et al. discuss NBTI effect in SRAM cell [10]. This work excludes analysis of dynamic input pattern on degradation. More recent work [11] measures degradation in reliability of 6T SRAM cell due to BTI. Kang et al. provide a thorough study on reliability issues related to SRAM, and discuss yield and other failure rates when considering SRAM arrays [12]. Both of these work do not propose any method to improve reliability. Yang et al. assess BTI impact on 8T SRAM cell [13] and power-gated SRAM cell [14]. But their work do not consider sequence of input bias pattern found in desktop computers. Kumar et al. investigate the SRAM reliability characteristics and propose the bit inversion techniques for SRAM based caches [3]. However, their technique is not applicable for timing critical components like register file as it may add extra delay for register access. Recent work [9] discusses limited benefits of previously proposed mitigation techniques for combinational circuits including an NBTI aware scheduling, lifetime awareness and Adaptive Body Biasing [15–17] However, our work targets register file, which has different a structure from combinational circuits.

At the architecture level, Abella et al. discuss the importance of NBTI in microprocessor design, and discuss a few methods for the critical structures of the microprocessor [1]. They consider the physical register file, and exploit idle cycles for recovering NBTI stress. Similarly, Siddiqua and Gurumurthi suggest using modified SRAM cells with *Recovery*

Boosting technique during unmapped period of physical registers [2]. Both of these work strongly depend on availability of idle cycles. In a tightly designed processor, where number of entries in physical register file are limited, available idle cycle period is shorter. In such architectures, benefits from idle period based recovery mechanism are smaller. Our technique handles NBTI stress itself in efficient manner rather than using idle cycles for recovery. Another similar work also targets the idle cycles in a superscalar out-of-order processor design [18]. Fu et al. propose microarchitecture changes in register file design to target PV and NBTI together [19]. However, their work improves access delay, not reliability.

DeBole et al. discuss NBTI degradation in a pipelined processor [20]. However, their work excludes NBTI impact on the register file. Wang et al. study temperature effect on NBTI degradation and propose input vector control techniques for reducing NBTI impact [21]. Tiwari and Torrellas propose aging-driven application scheduling to hide aging due to NBTI [16]. Khan and Kundu propose changing operating frequency and supply voltage at run-time to improve processor reliabilty against NBTI [22]. All of the above work target timings delays due to NBTI, not the noise margin.

Chapter 3

Background

SRAM cells are widely used in modern processors to implement register files and on-chip caches. SRAM cells have several key performance characteristics, which have widely varying effect with NBTI aging. Table 3.1 gives an overview of the aging effects on SRAM performance metrics [11, 12, 23, 24]. Apart from read and write related metrics, SNM is the critical reliability metric in an SRAM cell. During the read operation, SRAM cells become most susceptible to failure as the SNM is reduced substantially [25]. The SNM during the read operation is also called the Read Noise Margin [26]. For the rest of this report, we measure the SNM during read operations in the SRAM cell.

As the table outlines, read and write delay shows minimal effect from NBTI aging (write delay shows a modest improvement). The detrimental effect of NBTI wearout is most prominent in the SNM and the Data Retention Voltage (DRV), which dictate the read stability failures and power savings potential, respectively. In this work, we focus on characterizing and mitigating SNM degradation, which determines the reliable operation in an SRAM-based register file. We illustrate this degradation using HSPICE simulation next.

3.1 SNM Calculation in a SRAM Cell

Figure 3.1(a) shows the standard 6-T SRAM cell. It consists of two inverters that store complementary values at all times. The WL line is enabled to write a value, while the BL line is used to carry data to be stored in the cell. The data is retained in the cell by turning off access transistors M5, M6. To read data, the word line is set high and the bitline value is retrieved.

The SNM is the measure of the minimum DC noise voltage that leads to the loss of the

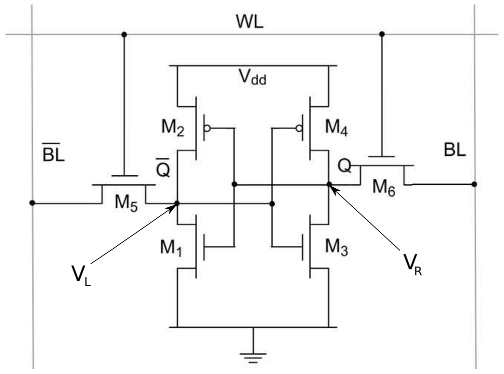
Table 3.1: Impact of NBTI aging on SRAM performance metrics.

Metrics	Description	NBTI Effect
Read Delay	Latency	Minimal degradation [23]
Write Delay	Latency	Modest improvement [23]
SNM	Minimum voltage causing bit flip	Significant degradation
DRV	Minimum supply voltage reqd. to retain data	Significant degradation [24]
Read Stability	Read failure rates	Negatively affected [11,12]
Write Stability	Write failure rates	Modest improvement [11,12]

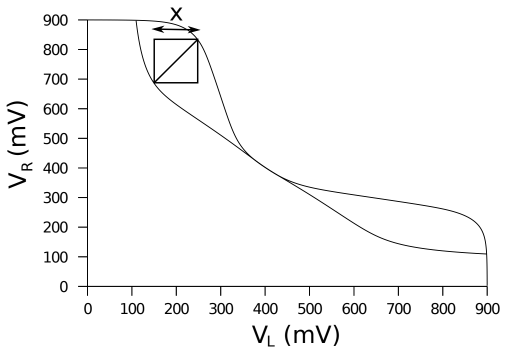
stored value. SNM value can be measured from the transfer characteristics of an SRAM cell. Figure 3.1(b) shows the butterfly curve of the SRAM cell during a read operation, which is used to find the SNM. The SNM equals to the side of the square nested between the two curves with the longest diagonal [26]. Therefore, the length X in Figure 3.1(b) represents the SNM of the SRAM cell.

3.2 NBTI Effect Measurement

When gate voltage of PMOS is negative, holes in inversion layer break Si-H bonds at oxide layer. It leads to increase in absolute value of threshold voltage of transistor.



(a) 6T SRAM cell.



(b) Transfer characteristics during read operation.

Fig. 3.1: 6T SRAM cell and butterfly curve.

This forms stress phase of NBTI. In recovery phase, gate voltage of PMOS is positive, annealing interface traps. Threshold voltage starts restoring to original value slowly. Similar chain of events occur in n-channel transistor (NMOS) with high-k metal gate due to electron trapping. NMOS is stressed when gate voltage is high, while recovery happens when input is low.

We model a standard 6-T SRAM cell in SPICE to measure its NBTI impact. The NBTI wearout leads to an increase in the threshold voltage of the transistors M2, M4 and M1, M3 respectively, which alters their transfer characteristics. As a result, the voltage difference between the cell nodes, or the noise margin, is reduced. We evaluate SNM at different time intervals by measuring this potential difference. We use a predictive NBTI model to determine the change in transistor threshold voltage due to NBTI [24]. Equation (3.1) provides closed form expression for upper bound of long-term threshold voltage change(ΔV_t) [8].

$$\Delta V_t = \left(\frac{\sqrt{{K_v}^2 \alpha T_{clk}}}{1 - \beta_t^{1/2n}} \right)^{2n} \tag{3.1}$$

Table 3.2 gives details of parameters used in Equation (3.1).

For the HSPICE simulations, we use the Predictive Technology Model (PTM) [27].

3.3 Input Bias Pattern Representation

Table 3.2: Parameters to estimate long-term threshold voltage change due to NBTI.

Parameter	Value
β_t	$1 - \frac{2\xi_1 t_e + \sqrt{\xi_2 C(1-\alpha)T_{clk}}}{(1+\delta)t_{ox} + \sqrt{C}t}$
K_v	$\left(\frac{qt_{ox}}{\epsilon_{ox}}\right)^3 K^2 C_{ox}(V_{gs} - V_t)\sqrt{C} exp\left(\frac{2E_{ox}}{E_o}\right)$
T_o	10^{-8}
C	$T_o^{-1}.exp(-E_a/kT)$
ξ_1	0.95
δ	0.5
$E_a(eV)$	0.49
$E_o(V/nm)$	0.335

Effects of NBTI are caused due to certain input values at transistors. Thus input pattern to transistor is primary driving factor for NBTI degradation. To understand input pattern better, we introduce a parameter, viz. One Bias Probability (OBP). OBP is probability of a transistor input to be at logic “1.” When OBP is close to “0.0” (0%), PMOS is stressed for most of the time. The NBTI recovery occurs when OBP of PMOS is at logic “0.”

3.4 SNM Degradation

Figure 3.2 shows the SNM degradation due to NBTI as a function of time across multiple technology nodes. There is a rapid deterioration in SNM in the first year, followed by more progressive deterioration. The deterioration is more pronounced at lower technology nodes. For example, 22nm technology node shows more than 40% degradation in SRAM reliability after 5 years.

In Figure 3.2, we assumed a OBP of 0.1. Essentially, OBP of SRAM cell indicates the period of time when cell stores logic value of one. The SNM degradation strongly depends on the OBP of the cell, and Figure 3.3 shows this relationship. Degradation is lowest when the OBP is exactly 0.5, as both the PMOS transistors experience uniform stress, resulting in least reliability degradation. Note that in the figure, we only show the OBP from one complementary PMOS. Similar SNM degradation has been also been shown in the recent past [3, 12].

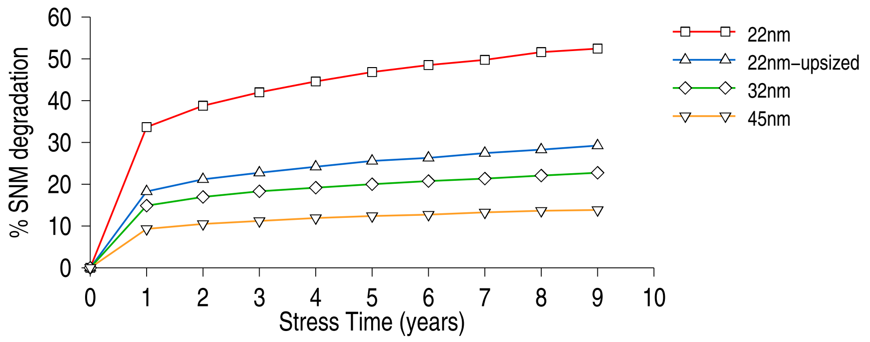


Fig. 3.2: Lifetime SNM degradation (NBTI).

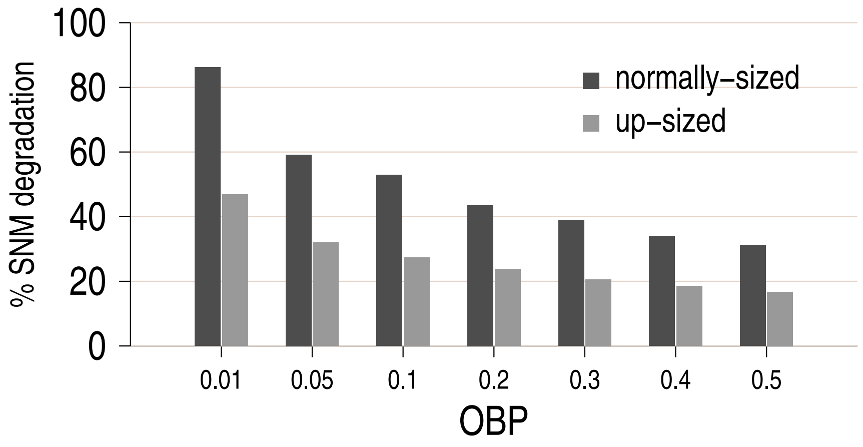


Fig. 3.3: Impact of OBP on SNM degradation due to NBTI (7 years, 22nm). The second bar indicates SNM degradation after upsizing transistors in the SRAM cell.

3.5 Impact of Transistor Sizing

Transistor sizing is one of the methods used to increase tolerance towards NBTI. Previous works have used transistor sizing for NBTI mitigation in combinational circuits [28,29]. In our work, we exploit up-sized transistors to improve the SNM of the physical register file (Chapter 5). The line marked with the up-triangles in Figure 3.2 indicates SNM degradation for such a cell. It can be observed that SNM degradation reduces by a large extent when compared to an SRAM cell without sized transistors (line marked with squares). Similarly, Figure 3.3 shows improvement in the SNM for all OBP values compared to the original cell.

3.6 Impact of Supply Voltage Scaling

The lost noise immunity of an SRAM cell can be improved by increasing the supply voltage of the circuit. Previous works have used the supply voltage scaling for increasing the NBTI tolerance of circuits [30,31]. Voltage scaling helps to mitigate NBTI effects, but it has a power overhead. Higher supply voltage results in the higher power consumption. Figure 3.4 shows improvement in SNM of an SRAM cell due to 20% voltage scaling.

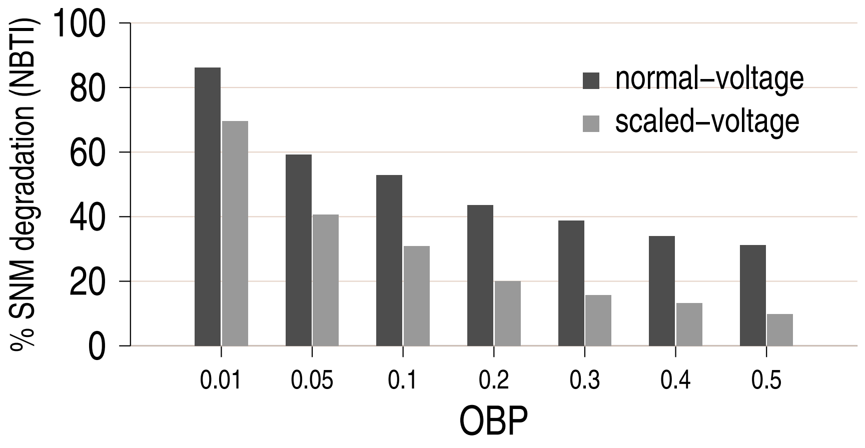


Fig. 3.4: Impact of OBP on SNM degradation due to NBTI (7 years, 22nm). The second bar indicates SNM degradation after 20% supply voltage scaling.

Chapter 4

NBTI Mitigation in Architecture Register File

In this chapter, we discuss NBTI impact on the architecture register file. We propose few techniques to balance NBTI stress in the architecture register file and present benefits of using new design.

4.1 OBP Characteristics of Applications

With SNM degradation profile as shown in Figure 3.4, it is now important to understand the stress induced in a real system. In this work our focus is on the register file. To get the program generated values, we simulate several SPEC CPU2006 benchmarks on a SPARC V9 architecture (see Section 5.4 for detailed methodology) [32]. Figure 4.1 shows the variation present in the OBP of four register groups (global, local, input, output) in the integer register file. There are eight 64-bit registers in each of these groups, and we show the average OBP across all the $8X64$ bits after a run of 100 million instructions.

Across a range of applications, the average OBP in the entire register file is 0.22. However, the key observation is the variability in OBP across the register file. For example, the lowest OBP is seen in the *local* registers for *gcc* benchmark, while the highest is seen in *hmmer* for the *input* register group. Even within an individual group of registers, applications show large diversity in the OBP. For example, the *in* registers show OBPs of 0.2 and 0.58 in *perlbench* and *hmmer*, respectively. Although this analysis is specific to the SPARC V9 ISA, we expect the general conclusion on variability across different programs to hold true for several other ISAs.

In the light of these wide variations in OBP across different applications, it is imperative to analyze the resultant effect of interleaving application characteristics. Indeed, in a real system, the Operating System (OS) schedules various applications to run on the processor,

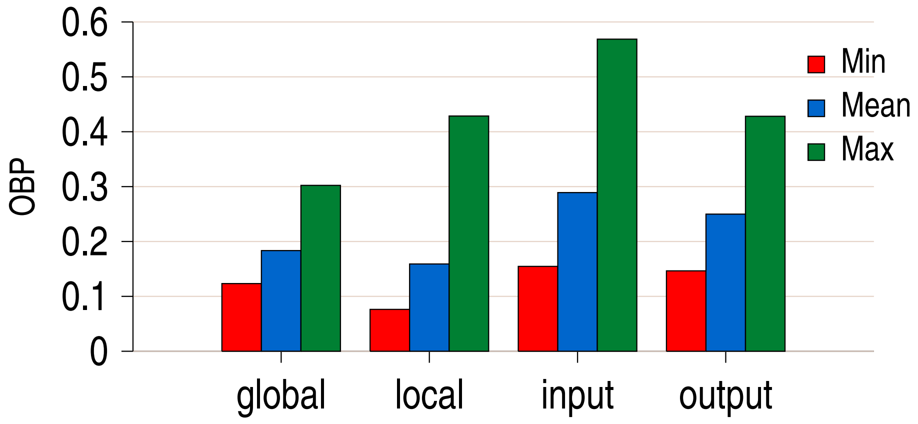


Fig. 4.1: Diversity in OBP among SPEC 2006 benchmarks.

thereby inducing a combination of stress patterns from individual programs on the register file. We investigate the impact of this behavior in Section 4.3, after briefly discussing the existing techniques to mitigate SNM degradation in the on-chip SRAM memory structures next.

4.2 Improving SNM Through Periodic Bit Inversion

To prevent SNM degradation in SRAM-based cache memories, Kumar et al. proposed a periodic cell flipping technique that aims to maintain bit OBP at 0.5 [3], whereas the average OBP across all registers is 0.22 (Figure 4.1). This technique inverts bits after regular time intervals to make sure each cell stores opposite values for half of the time. During the period of inversion, input data is inverted and stored while the outgoing data is sent back in inverted form. This simple technique works well in caches where bit values are less frequently changed. We refer to this scheme as INV in this work. The bit patterns in a register file, however, show widely varying OBP when we consider interleaving of program characteristics. We rigorously analyze such use scenarios next.

4.3 Impact of Application Interleaving

In this section, we investigate the impact of running multiple applications in arbitrary sequences on the SNM degradation.

In a modern desktop computer, the OS schedules different applications for small time epochs. Consequently, register bit patterns from different programs are interleaved, resulting in random sequences of bit values stored in the register SRAM cells. To investigate the impact of this real world scenario, we collect the one bias probability of every single register bit seen in various phases of a program. Different phases from different programs are then interleaved to create a sequence of realistic execution. We separate each benchmark in multiple 10ms phases, which is the typical OS scheduling quantum.

Figure 4.2 shows a pictorial representation. After collecting the OBP of each phase (α_0, α_1, α_2,...), we evaluate the impact of the INV scheme by complementing the OBP of alternating schemes. In any such sequence, the degradation of the entire register file will be dictated by the worst case OBP among all the register bits in the processor. The overall OBP after using the inverter technique is given by:

$$OBP_{INV} = 1/N * (\alpha_0 + 1 - \alpha_1 + \alpha_2 ...). \tag{4.1}$$

4.3.1 Limitations of Periodic Bit Inversion

Periodic bit inversion works well when the OBP remains more or less steady, as is expected within a single application. So when values of α_0, α_1, and so forth are close, the inverter technique gives an overall OBP of 0.5, completely negating their inherent bias.

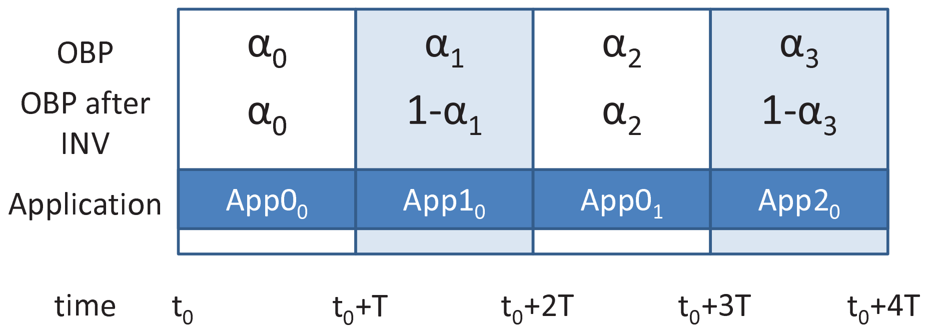


Fig. 4.2: Register bit level stress from sequence of application phases. We use 10ms as the time period T, a typical scheduling quantum in modern operating systems, in our analysis.

However, if the variance of OBP in the sequence is high, the resultant OBP starts to diverge away from the optimal 0.5. Consider, a situation when individual OBP are given as (0.1, 0.9, 0.2, 0.8, 0.1,..). In this case, the variation is high and overall OBP is much smaller than 0.5.

4.3.2 Results of Application Sequences

Manifestation of NBTI degradation in the chip-level performance occurs over a long period of time, typically 5–10 years. Clearly, it is impractical to run a simulation for such a long period. Therefore, to estimate the impact of arbitrary application sequences over such a long time, we followed the following methodology. Our goal here is to determine the worst case OBP in the register file, as degradation in a single bit dictates the overall degradation of the architectural register file.

To realize this goal in an experimental setup, we want to understand the impact of application interleaving on the resultant OBP for a typical day in the entire aging period. Subsequently, we assume that such days dominate throughout the aging period, determining the OBP for the entire aging period. However, even a full day's simulation can take multiple years in an architectural simulation tool. To resolve this issue, we first analyze 15 different benchmarks from the SPEC CPU2006 suite. Since in the presence of other runnable applications, a typical case in the real life, the OS will schedule an application for a pre-defined scheduling quanta (e.g., 10ms), we collect the OBP of several consecutive runs of 10ms in each benchmark. Finally, we randomly combine these phases to construct a typical application interleaving scenario of a whole day from these benchmarks. Since these combinations can be done in a large number of ways, we randomly select 2000 of such combinations. Within each combination, we analyze the worst case OBP across all register bits using the INV technique. Figure 4.3 shows a scatter plot from this study.

Clearly, when we consider application interleaving, periodically inverting the bit values is unable to bring the resultant OBP close to the best possible value. The median OBP in the sample is 0.131. For more than 30% of the examined execution sequences, we notice a OBP lower than 0.05, indicating severe SNM degradation (Figure 3.3). In general, the

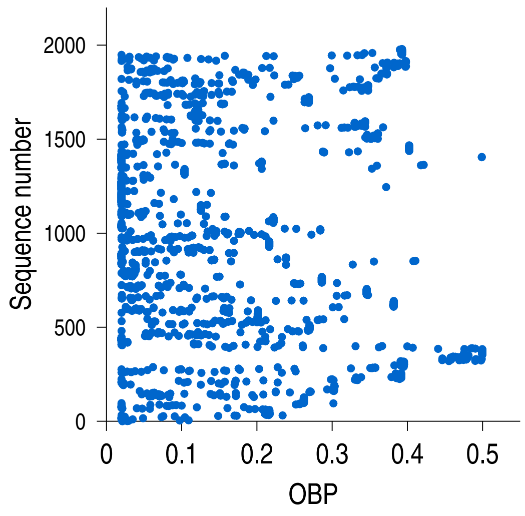


Fig. 4.3: OBP with periodic inversion on application sequences (0.5 is optimal).

uncertainty of the resultant OBP is high, with a standard deviation of 0.142. Since, INV of SRAM bits is unable to adapt to the fluctuating program characteristics, it cannot achieve the desired OBP for several possible application interleaving patterns.

4.4 Micro-Architecture Technique

We now propose two micro-architecture techniques to balance the NBTI stress across the register file. The first technique targets irregular register usage by altering the register decoding, while the second targets the bias in program values. Together, these techniques reduce the bias and variability of the OBP in SRAM arrays, along both the column and rows of a register file. We describe these two techniques and their combination (Sections 4.4.1, 4.4.2, and 4.4.3), and then discuss their implementation in a pipelined microprocessor in Section 4.4.4.

4.4.1 Register Rotation (RR)

Each register file has rows and columns of 6T SRAM cells to store register values. A particular architectural register uses one such row where bits are stored in column cells. Traditionally, the same physical SRAM cell is used for storing the value of a particular register throughout the processor's lifetime. This static mapping between architectural register and the physical SRAM cells lead to a high variance in OBP seen across the entire register file. Our goal is to eliminate these static links between registers and SRAM cells by dynamically changing the register decoding scheme.

For the ease of illustration, Figure 4.4 shows a simple 8-bit register file with eight registers. Subsequent evaluations are carried out on 64-bit registers in SPARC V9. In order to balance the OBP, we introduce a barrel shifter between the address decoder and the memory cells. The barrel shifter rotates select lines after regular time intervals. For example, after a time interval T, register 0 is mapped to the row 1. Over a long time span, different rows of memory cells are used for register 0. Consequently, both high variance and inherent bias can be negated using this technique.

The process of shifting the select line is repeated after fixed intervals of time. During such cycles, the *shift count* is incremented by one and fed to the barrel shifter. Figure 4.4 shows the change in shift count after regular intervals of time period T. Repetitive shift operations results in a complete rotation, which ensures that all eight rows of memory cells are used for storing register 0.

4.4.2 Bit Level Rotation (BR)

Bit level rotation performs the same operations, but targets column cells rather than the row cells. The barrel shifter in the write port rotates the input data and stores it into memory cells. Therefore, when shift count is 1, bit 0 is stored in column 1. The bit 0 gets shifted to the left as the count goes up. After N (width of registers) rotations, bit 0 has used every column of cells to store its value.

4.4.3 Combined Register and Bit Level with INV (RBR+INV)

We can combine both register level and bit level techniques to simultaneously distribute the variance and inherent bias across the entire SRAM arrays in the register file. Figure 4.5 shows the sequence of operations. However, as the values are inherently biased towards

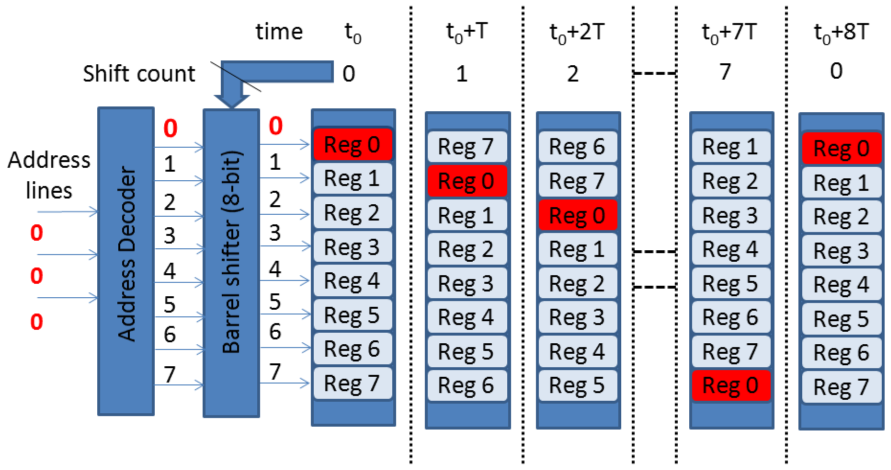


Fig. 4.4: Register level technique. The barrel shifter dynamically rotates the select line by *shift count*.

zero, these techniques will not bring the OBP to the optimal 0.5. Therefore, we combine INV with our rotation techniques.

4.4.4 Implementation

Our techniques change the physical location of architectural registers. In a pipelined microprocessor there are two specific steps before such a change can be possible: (a) pipeline flush; and (b) update values to the new location.

Pipeline Flush: Correct instruction execution depends on the association of appropriate values to the respective registers. Consequently, it is impossible to alter the register decoding (or change the bit position interpretation) in presence of in flight instructions. Before we can change the decoding scheme or allow bit positions to interchange, we must ensure that the pipeline is flushed, so that old and new decoding is not mixed during the execution of a single instruction.

Value Update: Before new instructions are fetched in the pipeline, we must also ensure that mapping alteration of the physical SRAM cells and the register bit position is hidden from the program. Therefore, we must update values in the registers as dictated by the change in decoding or bit position interpretation. For example, when rotating *Reg0* to

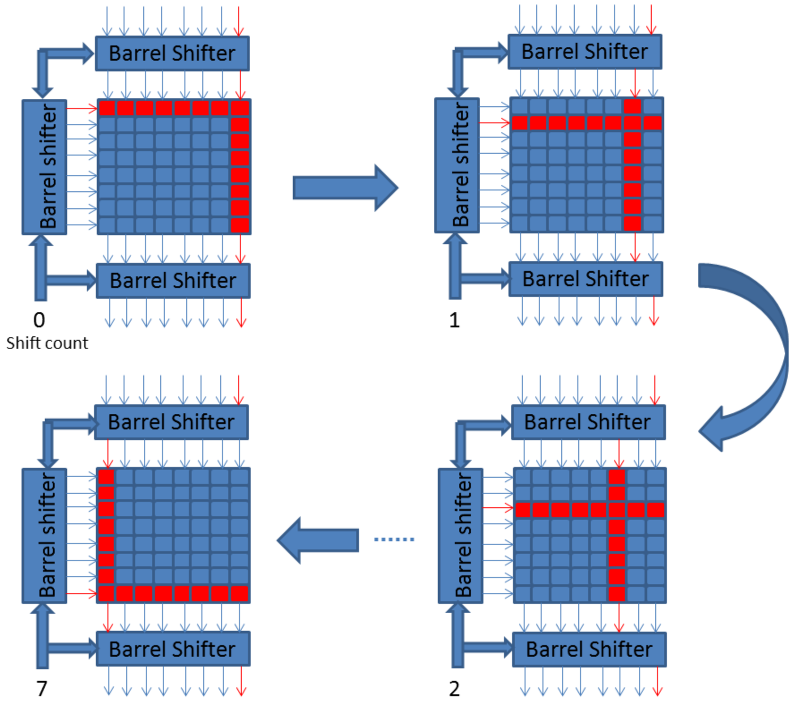


Fig. 4.5: Register and bit level technique.

Reg1, we must write the value of *Reg0* to *Reg1*, *Reg1* to *Reg2*, and so on.

Similar steps are also necessary when using periodic inversion of values (INV scheme). Although pipeline flush is inexpensive in an in-order simple pipeline (few clock cycles only), *Value Update* can be expensive. To avoid penalty from the latter, we perform periodic mapping alteration during the OS induced *context switch* operation. Since the architectural register state is saved and restored at the boundary of context switch, we can simply perform the mapping change once the register state is saved from one thread. Before the register thread from another thread is restored, we alter the mapping. All subsequent operations are carried out in this new mapping, till OS induces the next context switch. Fundamentally, performing these operations at the boundary of context switch allows these mechanisms to have negligible clock cycle penalty, while maintaining complete transparency from the high level software. For all schemes, we ignore penalty from these operations.

4.5 Results

We present the resulting OBP from three different techniques: INV (periodic bit inversion), and our proposed techniques of register and bit level rotation (RBR), and RBR with INV (RBR+INV).

Figures 4.6(a) and 4.6(b) show the OBP scatter plot of sequence executions with RBR and RBR+INV, respectively. Identical execution sequences were shown earlier in Figure 4.3 for INV. Compared to the INV, RBR substantially reduces the variation in OBP seen for these execution sequences. However, since no bit inversion is used, the resultant OBP is tightly distributed around the mean OBP of 0.22 across all programs (see Figure 4.1). Figure 4.6(b) demonstrates that combining the bit inversion with our technique further improves the OBP, and effectively pushes it towards the optimal 0.5.

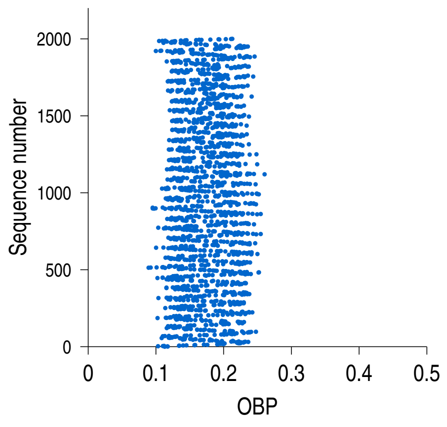
Table 4.1 gives the summary of these results. Since conservative designs considering the absolute worst case often leads to over-design, we show the 10th percentile OBP in this table. The 10th percentile OBP indicates that all but 10% of the random sequences have better OBP after using the respective techniques. The SNM degradation shown considers the 10th percentile case. We also show the uncertainty measurements as the standard deviation of the OBPs seen for these execution sequences. For the 10th percentile case, our technique is able to reduce the SNM degradation by 2.2X. In addition, the uncertainty is lowered by 14X, leading to a substantially robust design.

4.6 Overhead Analysis

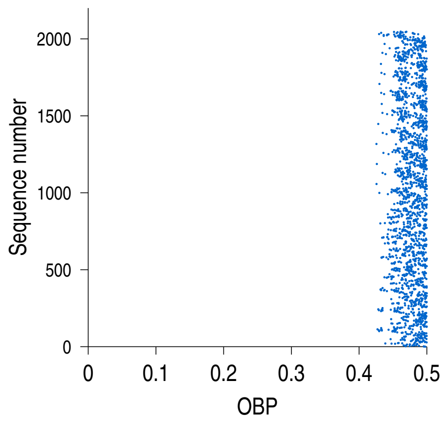
We now present the overhead analysis of various schemes discussed in this work.

Table 4.1: Comparative OBP and SNM degradation.

Scheme	Median (OBP)	10th Perc. (OBP)	SNM Degra-dation	STDDEV
INV	0.131	0.02	31.3%	0.14
RBR	0.17	0.13	20.7%	0.03
RBR+INV	0.48	0.46	14.4%	0.01



(a) RBR.



(b) RBR+INV.

Fig. 4.6: Scatter plot for sequences with different schemes.

4.6.1 Methodology

To estimate the overhead of various schemes, we create Verilog description of the register file, along with different schemes. We verify the functionality of the Verilog using the ModelSim simulation tool. Subsequently, we synthesize the hardware using Synopsys Design Compiler and a 45nm TSMC library, which consists of three different threshold voltages (V_t). We synthesize different hardware schemes for the same target latency, by the appropriate selection of threshold voltages and gate sizes in the circuit. Thus, we do not allow any access latency overhead, but evaluate the area and power overheads for maintaining identical performance.

4.6.2 Results

We find that the INV scheme has marginal area overhead of 3.4% over the register file without any NBTI mitigation technique, but has a modest power overhead of 18%. Since we want to compare the overhead between different NBTI mitigation techniques, Figure 4.7 shows the overhead of various schemes proposed in this work, compared to the INV scheme. RR incurs lower overhead in area and power, as the necessary hardware component for register level rotation is smaller than performing bit-level inversion in a 64-bit register file. The best scheme (RBR+INV) shows an overhead of 8.8% in area, and 7.8% in power, compared to the INV scheme.

Next we calculate overall overhead of modified register file with respect to entire processor core. From previous work, we observed that register file can take approximately 10% of the total area and 20% of the total power of processor core. Assuming these values, we find that our best scheme (RBR+INV) has small area and power overhead of 0.91% and 1.84%, respectively.

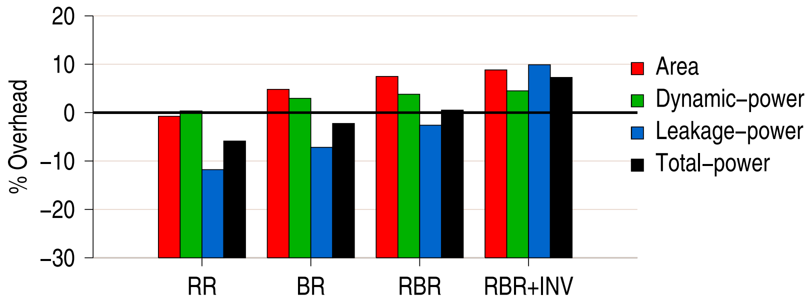


Fig. 4.7: Overhead of various schemes compared to INV.

Chapter 5

NBTI Mitigation in Physical Register File

In this chapter, we discuss NBTI impact on the physical register file. We study the NBTI stress generation in the physical register file at instruction granularity. Based on our observation, we propose modifications in the physical register file to efficiently handle NBTI stress.

5.1 Instructions and NBTI stress

In this section, we study the relationship between the bit patterns of values computed from an individual instruction and its impact on NBTI stress. We find that instructions often have highly predictable NBTI stress patterns. This observation opens up new opportunities to design NBTI-aware micro-architectures.

5.1.1 Bias Predominance

Typically, values stored in a register file are narrow-width numbers [33–35]. Hence, there is a large number of bit positions at logic 0, which can potentially lead to high NBTI stress. Similarly, there is a substantial number of instructions producing outputs capable of generating NBTI stress. To quantitatively measure the NBTI stress generated by an instruction, we look at the instruction's output bias. When a majority of output bits are at logic 0, the output can generate high NBTI stress. We define a parameter to indicate the output's bias towards logic 0, viz. Zero Predominance (ZP). We formally define ZP as follows: the ZP of an instruction is 1 when more than 75% of its output bits are at logic 0. ZP indicates an instruction's ability to produce NBTI stress in register file.

Figure 5.1 shows the presence of a large number of bit positions in a register file undergoing high NBTI stress. We plot the distribution curve of each bit position against its

OBP. Figure 5.1 demonstrates bias distribution curves for two benchmark programs, widely varying in their characteristics. The perlbench has its highest peak near 0.0, meaning it has the largest number of bits at logic 0. On the other hand, bzip2 has the smallest number of bit positions at logic 0. Relatively, the number of bit positions at logic 1, is small for all programs. On average, 57% of the bit positions are always at logic 0 and 18% of bits have OBP close to 0.5. In other words, 57% of SRAM cells in a register file might store logic 0 during their entire lifetime. Above observation proves that a large number of SRAM cells in the register file can potentially suffer from unbalanced stress and hence high SNM degradation.

Figure 5.2 shows a plot of zero bias predominance for SPEC CPU2006 benchmark programs. Each bar indicates the percentage of instructions that compute a new value with the ZP of 1. Figure 5.2 shows that perlbench has the highest number of instructions with ZP equal to 1. For perlbench, 71.16% of the total number of dynamic instructions produce outputs with a ZP of 1. On the other hand, bzip2 has the least number of instructions with ZP equal to 1. On average, 37% of instructions have the potential to create high NBTI stress in a register file, among all of the instructions that compute a new value.

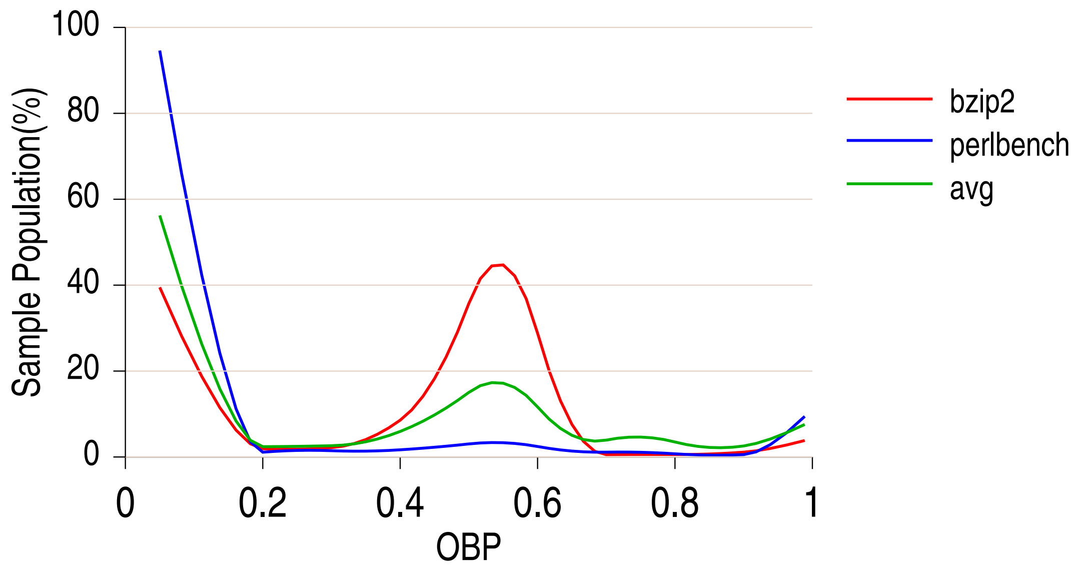


Fig. 5.1: Bias distribution for output bit positions.

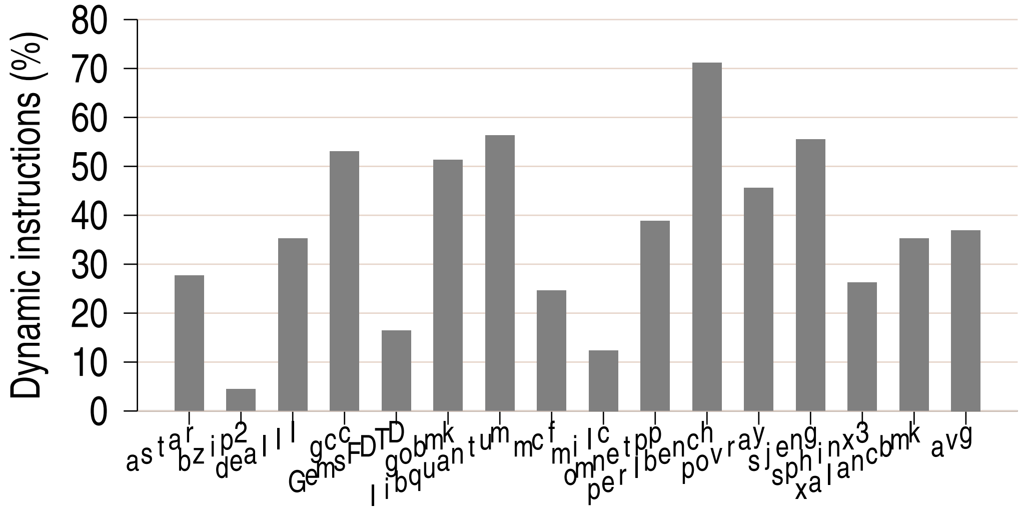


Fig. 5.2: Zero predominance of instructions that compute new value.

5.1.2 Bias Predictability

Figure 5.2 shows that a large number of instructions can produce high NBTI stress in the register file. Detection of such instructions before execution can be used to efficiently handle NBTI stress. In this subsection, we discuss the predictability of instructions to generate large stress in a register file. To predict the bias predominance of any dynamic instruction, we look at the output bias probabilities of a corresponding static instruction. When the OBP of a bit position is 0.0 or 1.0, it implies that the value has not changed during the entire execution time. Hence, the value of such a bit position is highly predictable. To measure the predictability of the complete instruction output, we count the predictable bits in the output. Based on the number of predictable output bits, we classify instructions into four categories.

Figure 5.3 plots the percentage of instructions with four levels of output bias predictability. Each bar in the plot is made up of four stacked components corresponding to different levels of the output bias predictability. The lowermost component indicates the set of instructions that produce output values with more than 75% of bit positions being predictable. In other words, more than 75% output bits of such instructions have an OBP

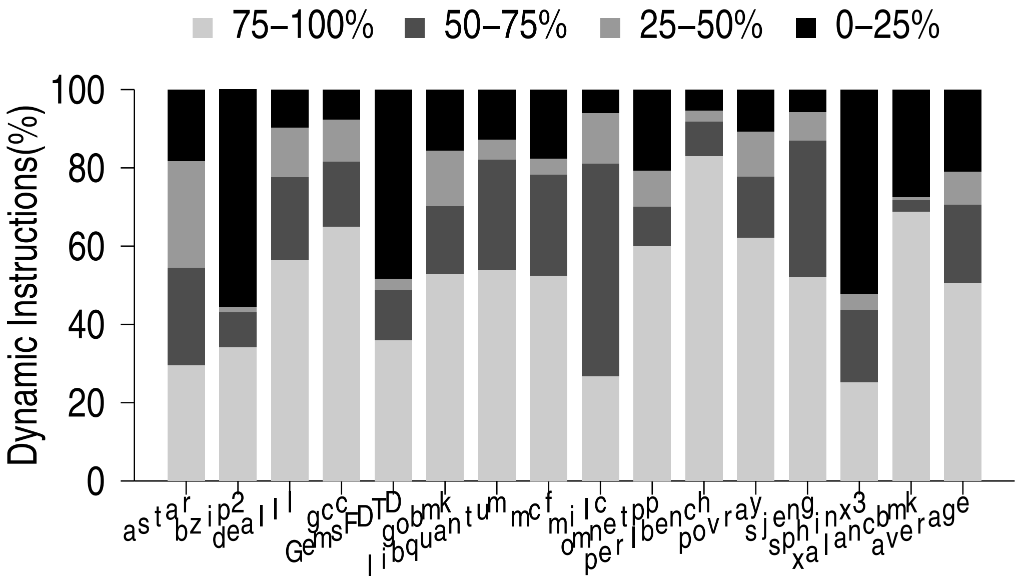


Fig. 5.3: Level of predictability for instruction outputs.

close to 0.0 or 1.0. The second component from the bottom indicates the set of instructions that have more than 50% predictable output bit positions. On average, 71% of dynamic instructions have more than 50% predictable bit positions. This observation proves that the output bias predominance of a large number of instructions is predictable, thereby instructions generating high NBTI stress can be predicted. Next, we develop a mechanism to identify and predict occurrence of stress generating-instructions.

5.2 Predicting Instruction Level NBTI Stress

In this section, we discuss key predictor designs and analyze the performance of each. We explore three specific designs: (a) Last Value Predictor (Section 5.2.1), (b) Bimodal Predictor (Section 5.2.2), and (c) NP Predictor (Section 5.2.3).

5.2.1 Last Value Predictor

From Figure 5.3, it can be observed that approximately 51% of the instructions produce values with constant bias. With this observation, we introduce the Last Value Predictor that predicts the output ZP to be the same as ZP of the previous instance. The Last Value Predictor stores the previous ZP value in the history table for each static instruction. As the ZP can have only two possible values, a single bit for each static instruction is enough.

The prediction accuracy of the Last Value Predictor is highest when instructions produce constant values. The Last Value Predictor faces misprediction each time the instruction output predominance is different than the previous instance. Hence, a single deviation from predominance pattern results in two mispredictions. Also, instructions producing a sequence of numbers cause mispredictions with the Last Value Predictor. Considering the above cases of mispredictions, we improve the predictor design by using two bits to store the predominance state.

5.2.2 Bimodal Predictor

With the bimodal predictor, rather than storing the actual ZP value in the history table, we store the state of bias predominance prediction. The state of bias predominance prediction is represented with the help of a 2-bit saturating counter. Figure 5.4 shows the state diagram for the bimodal predictor. Initially, the counter is in the *strongly zero predominant* state. It remains in the same state as long as the ZP of each static instruction's instance is high. When an instance produces an output with a low ZP, its state is changed to *weakly zero predominant*. Successive instances of instructions with low ZP values will change the state to *strongly zero non-predominant*. The *strongly zero non-predominant* state corresponds to the pattern where instances of static instructions produce less than 75% of output bits with logic 0.

The bimodal predictor adapts better in situations when the ZP value is alternately changed. Also, the bimodal predictor can tolerate a single deviation from bias predominance pattern without misprediction. However, the bimodal predictor results in two mispredictions before adapting to a new pattern.

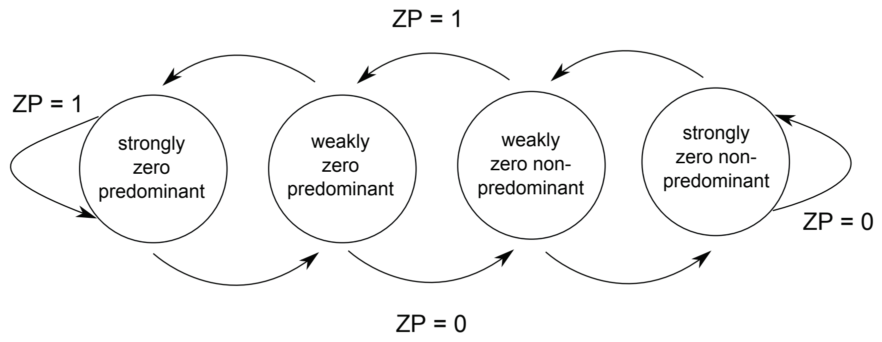

ZP = 1
ZP = 1
strongly zero predominant
weakly zero predominant
weakly zero non-predominant
strongly zero non-predominant
ZP = 0
ZP = 0

Fig. 5.4: Bimodal predictor state transition.

5.2.3 NP Predictor

The above predictors store the prediction state of all encountered instructions, irrespective of their ZP value. The NP (Non-Zero Predominance) predictor keeps track of instructions with low ZP only. As a limited number of instructions are tracked, fewer entries are stored in the NP predictor than the last value and bimodal predictors. For the NP Predictor, a misprediction occurs when an instruction having an entry in the predictor table generates an output value of low ZP.

5.2.4 Predictor Performance

Figure 5.5 presents the misprediction rate of the three predictors, each having 8k entries. It can be seen that the misprediction rate of the NP Predictor is the lowest of the three predictors. As the NP Predictor records instructions with low ZP only, there are fewer collisions than the last value and bimodal predictors, thereby increasing accuracy. On average, the misprediction rate of the NP predictor is half of the other two predictors.

5.3 Minimizing NBTI Degradation in the Register File

After successful prediction of the instructions generating high NBTI stress, we use this information to minimize NBTI impact in the register file. In the following subsections, we describe proposed modifications in the physical register file for improving reliability.

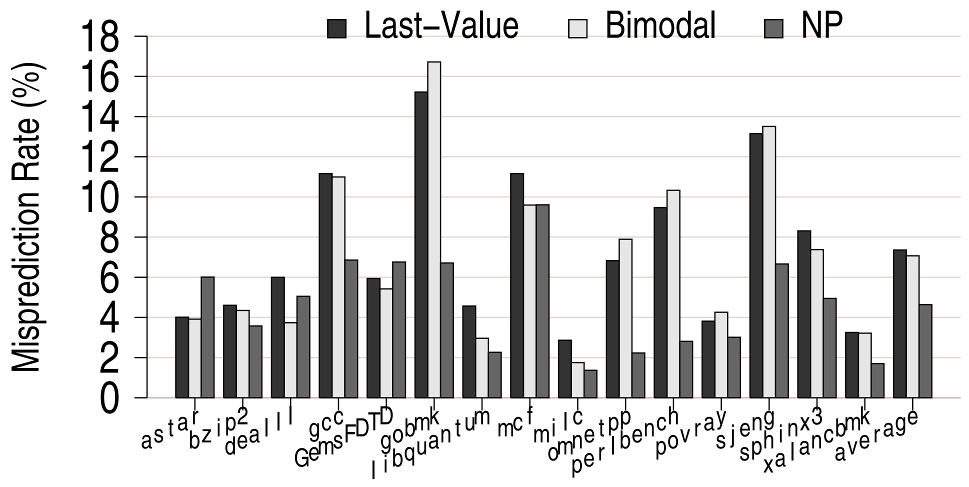


Fig. 5.5: The misrediction rate.

5.3.1 Design Overview

We split the register array into two banks, each having an equal number of registers. One of the register banks is exclusively used for allocating registers for outputs of the zero predominant instructions. As most of the zero predominant outputs are narrow-width, we compress register widths to 16-bits. The reliability of the narrow-width register bank is further improved by using up-sized transistors. The second register bank is used for the remaining instructions which produce non-zero predominant outputs.

We change the register allocation policy to allocate registers based on the predicted ZP value of an instruction. We handle special cases of mispredictions by performing a remapping of the physical registers. Section 5.3.3 describes modification in the decode and execute stages to handle mispredictions in detail.

5.3.2 Register File Modifications

In this subsection, we describe proposed changes in the structure of the physical register file.

Banked Register File

As mentioned earlier, the SRAM cells storing logic 0 (or 1) for large period of time undergo high NBTI stress. The SRAM cells of more significant bits in a register file show similar behavior. Compressing such outputs can reduce the number of bits at logic 0 without losing important information. Based on this observation, we divide the register array into two banks with different register widths. Configuration (c) in Figure 5.6 shows the new register file with two banks of variable widths. The first bank consists of 64-bit registers, while the second bank consists of compressed 16-bit registers.

NBTI Tolerant Bank

By compressing the widths of zero predominant outputs, we get rid of a large number of bits storing logic 0 and save a substantial amount of NBTI stress. However, there could still be many bits at logic 0 in the compressed output. These bit positions can further reduce overall reliability of a physical register file. To increase the noise margin of a narrow-width bank, we use up-sized transistors for NBTI tolerance. As seen in Figure 3.2, the effective SNM degradation of SRAM cells employing up-sized transistors is smaller than that of SRAM cells with normally sized transistors. Therefore, introduction of up-sized transistors improves the overall reliability of entire register file.

Above two changes remove uniformity from the original physical register file structure. The new physical register file has some non-uniformity due to presence of banks with different widths. This assymmetry can result in path overlaps during the routing phase. However, the number of such overlaps would be small as register file is split into two banks only. Hence, these overlaps can be easily removed.

Configuration (b) and (d) in Figure 5.6 show tolerant register banks in gray. Benefits of register file banking and up-sized transistors are discussed in Section 5.5. Next, we describe the modified register allocation policy and special cases of mispredictions in detail.

5.3.3 Modified Register Allocation Policy

We require modifications in the register allocation policy as one of the banks is exclusively used for zero predominant outputs only. Figure 5.7(a) outlines the new register allocation process. After predicting the ZP value for the given instruction, a register from

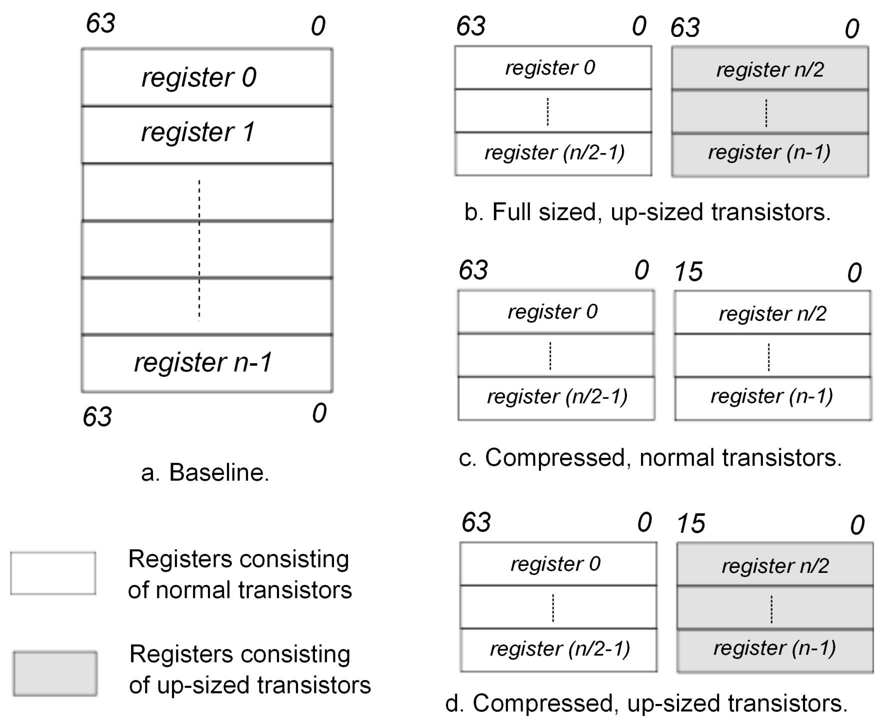
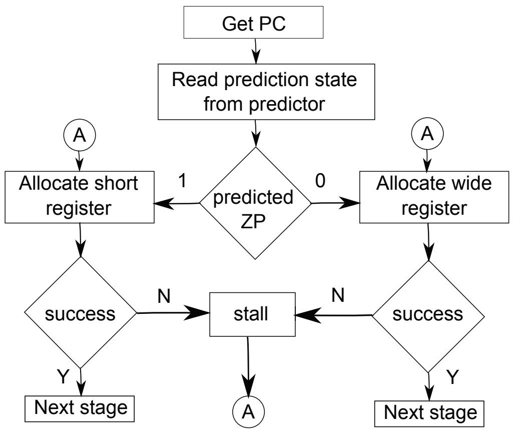

63
0
register 0
register 1
register n-1
a. Baseline.
register n/2
register (n/2-1)
register (n-1)
b. Full sized, up-sized transistors.
15
c. Compressed, normal transistors.
d. Compressed, up-sized transistors.
Registers consisting of normal transistors
Registers consisting of up-sized transistors

Fig. 5.6: Various configurations of register file.

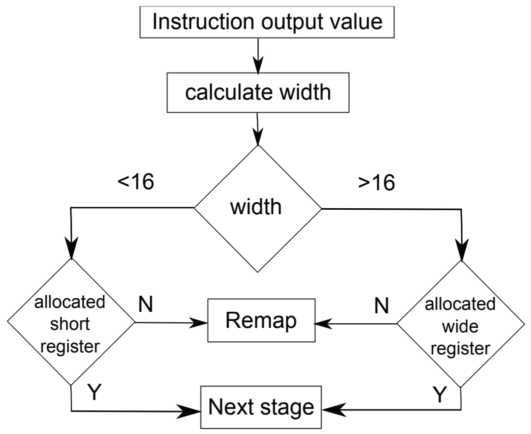
one of the banks is allocated. If the ZP is high then a short register from the narrow-width bank is allocated for destination. Similarly, a wide register is allocated for the instruction with a low ZP. During certain program phases, when there is a high demand for registers from one of the banks, register allocation may fail. In such a situation, we stall the given instruction in the decode stage and wait for registers to get free.

Based on the predicted value of zero predominance, a short or wide register is allocated to the given instruction during the decode stage. If the predicted value turns out to be false, then a misprediction occurs. When an instruction is mispredicted to be zero predominant and its output is not a narrow-width value, the allocated destination register is not wide enough to contain the output. For maintaining the correctness, a wide register from the first bank must be re-allocated for a non-zero predominant instruction. Figure 5.7(b) shows the flow of events for remapping during execution stage.

With the modified register file, performance of the processor can suffer in two situations. First, performance may be affected due to remapping during the execution stage,



(a) Flow of events during decode stage.



(b) Flow of events during execution stage to handle bank mispredictions.

Fig. 5.7: Modifications in register allocation policy.

as explained earlier. In another situation, performance is lost when register allocation fails during the decode stage. If no free registers are available in the requested register bank, the instruction is stalled. We found that the loss in IPC due to the above cases is small. We observed that average performance overhead is less than 1.5%.

5.4 Methodology

In this section, we discuss experimental methodology used in our work.

5.4.1 Architectural Simulations

For investigating register file configurations and its impact on performance, we use a full-system simulator built on top of the Wind River SIMICS [36]. For our experiments, we use the SPARC V9 ISA. However, we use our own detailed timing model to enforce timing characteristics of a 4-wide superscalar out-of-order core. Our modeled processor has a register file structure similar to SPARC V9. The architecture register file contains 160 windowed registers. We have implemented MIPS R10K style register renaming [37] with a physical register file of 224 registers. Various register file configurations shown in Figure 5.6 are explained in Section 5.5.

We use several SPEC CPU2006 benchmarks on a Solaris 9. We use the three most representative phases from these SPEC benchmarks in our study, extracted using the SimPoint toolset [38].

5.4.2 NBTI Effect Measurement

When the input to PMOS transistor is low, holes in the inversion layer break Si-H bonds at the oxide layer. This phenomenon leads to an increase in the absolute value of the threshold voltage of the transistor (*stress phase*). When the input to a PMOS transistor is high, the threshold voltage slowly starts restoring to the original value (*recovery phase*).

We model a standard 6-T SRAM cell in HSPICE to measure NBTI impact. The NBTI wearout alters the transfer characteristics of cross-coupled inverters, decreasing the noise margin. We evaluate the SNM at different time intervals by measuring this potential difference. We use a predictive model to determine the change in transistor threshold voltage due to NBTI [24, 27].

5.4.3 Methodology for Area and Power Estimations

For finding the savings in area and power of the modified register file, we describe structures in Verilog. We synthesize the hardware using the Synopsys Design Compiler and the 45nm TSMC library. We synthesize various register file configurations for fixed latency and obtain the area and power estimation from this synthesized Verilog. We could not produce results for 22nm and 32nm technology nodes as we do not have access to lower technology node libraries in our academic setup.

We use the CACTI 6.0 [39] tool to perform area and power analysis of the predictors. We compute area and power values for the 45nm technology node so that the overall savings at 45nm can be known.

5.4.4 Recovery Boosting Technique

Siddiqua and Gurumurthi propose the recovery boosting technique to extend the recovery of SRAM cells during the idle cycles of the physical registers [2]. This technique uses modified SRAM cells where the PMOS transistors undergo recovery phase during the invalid period. To find effectiveness of the recovery boosting technique, we model a modified SRAM cell in HSPICE. Based on the length of idle cycle period and bias probabilities of bit positions, we calculate average SNM of the entire register file. Success of the recovery boosting technique strongly depends on the length of idle cycle period. Longer the idle cycle period, better is the SNM of the register file. However, the length of the idle cycle period varies from processor to processor. A tightly designed processor has shorter idle cycle period. Figure 5.8 shows average idle cycle percentage for three different configurations of the physical register file. In Figure 5.8, register file configuration *conf1*, *conf2*, and *conf3* have 200, 224, and 248 physical registers, respectively. The number of logical registers in each configuration is same, i.e. 160.

5.5 Results

In this section, we present the results of improvement in SNM and power efficiency due to modifications in the register file. We compare our results with the recovery boosting technique. We experiment with various configurations of the physical register file and show the results for SNM improvement in Figures 5.9-5.11. Then we discuss the savings in area and power due to our proposed technique.

We evaluate five different configurations of the register file. All results are shown with respect to *Baseline* configuration in Figure 5.6.

- *Baseline:* This refers to the traditional design of a physical register file with 64-bit wide registers only. Refer to configuration (a) in Figure 5.6.
- *comp,norm:* This refers to a compressed banked register file design where one of the banks has 16-bit wide registers only. Refer to configuration (c) in Figure 5.6.
- *rec:* This refers to the recovery boosting technique. The idle period results for our processor configuration are shown in Figure 5.8.

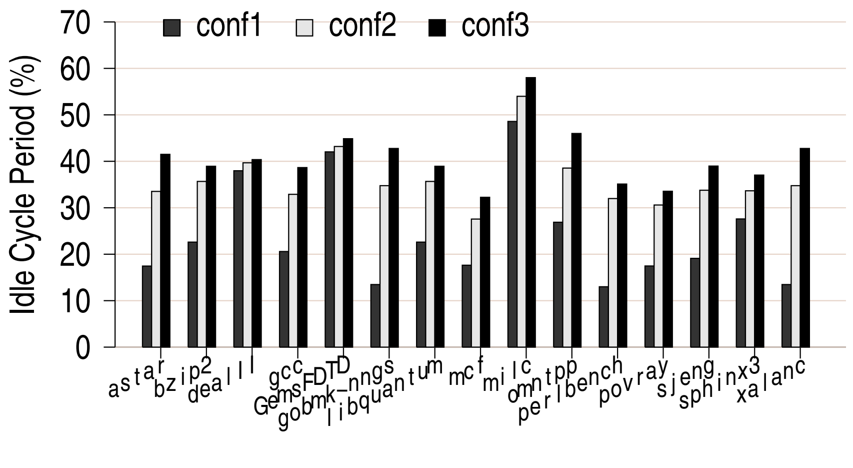


Fig. 5.8: Idle cycle period for various register file configurations.

- *full,up:* This refers to a banked physical register file where all registers are 64-bit wide only. One of the banks is made up of up-sized transistors. Refer to configuration (b) in Figure 5.6.

- *comp,up:* This refers to a compressed banked register file design where one of the banks has 16-bit wide registers only. One of the banks is made up of the up-sized transistors. Refer to configuration (d) in Figure 5.6.

5.5.1 Results for SNM Improvement

Figure 5.9, 5.10, and 5.11 show the average percentage improvement in the SNM of register file configurations for the 22nm, 32nm, and 45nm technology nodes, respectively. The first bar in the cluster indicates 9-12% (32nm) improvement in SNM with compression of one register bank to 16-bits. The recovery boosting technique results in 18-26% SNM improvement. For *full,up*, the SNM increase is highest among all configurations. The last configuration *comp,up* gives a smaller improvement in SNM compared to *full,up*, but with the highest savings in power and area. Ideally, we expected the SNM to increase from *full,up* to *comp,up* as the number of SRAM cells of higher significant bits are reduced. But after careful analysis, we found that the SNM of significant SRAM cells with up-sized transistors is more than the average SNM of the first bank. Hence, SRAM cells with up-sized transistors for significant bits improve the average SNM of the entire register file.

On average, *comp,up* shows an SNM improvement of 20%, 32%, and 125% for the 45, 32, and 22nm technology nodes, respectively.

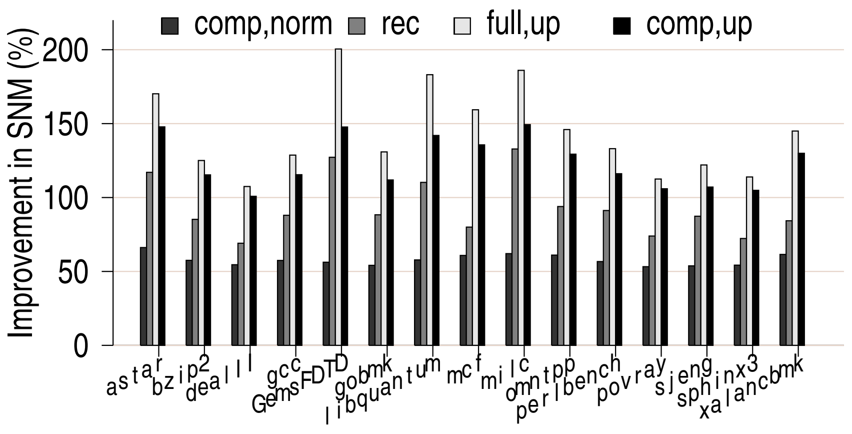


Fig. 5.9: Percentage improvement in SNM with respect to physical register file with a single bank and nominal transistors for 22nm technology.

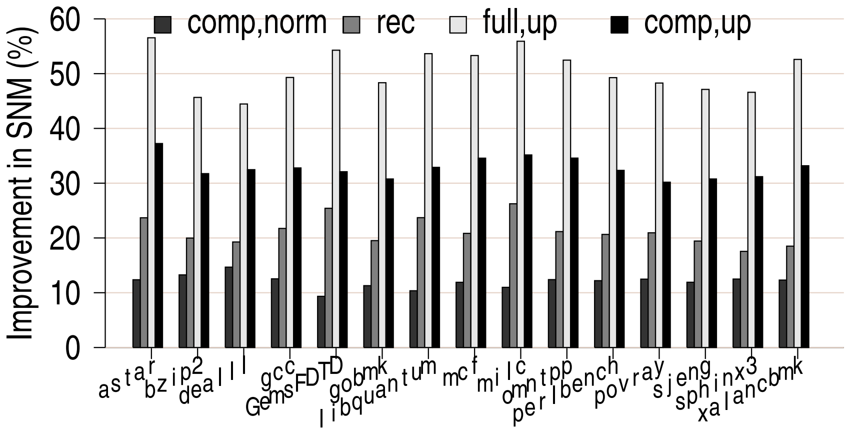


Fig. 5.10: Percentage improvement in SNM with respect to physical register file with a single bank and nominal transistors for 32nm technology.

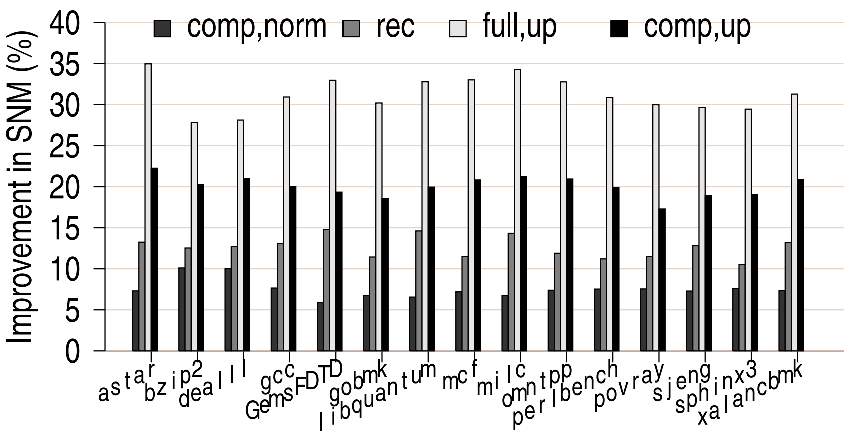


Fig. 5.11: Percentage improvement in SNM with respect to physical register file with a single bank and nominal transistors for 45nm technology.

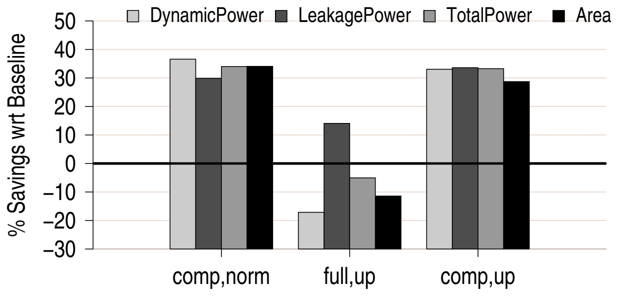
5.5.2 Area and Power Comparison

Our technique improves SNM by compressing registers without losing the functional correctness. Effectively, the modified register file has fewer SRAM cells. Reducing the size of hardware results in savings in area and power. Figure 5.12(a) shows percentage savings for area and power for our proposed techniques with respect to baseline.

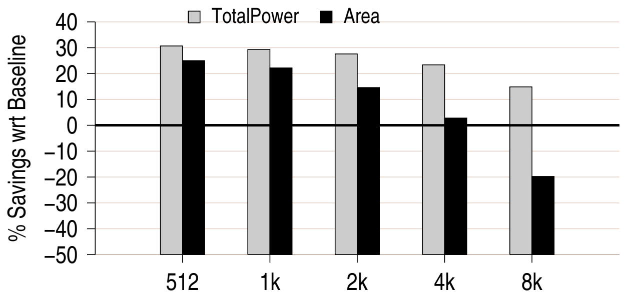
Configuration *comp,norm*, where the second register bank is compressed to 16-bit only, provides 34.5% savings in overall area. As the number of SRAM cells are reduced, both dynamic and leakage power decrease. Overall savings in dynamic, leakage and total power are 37%, 29.9%, and 34.4%, respectively.

When transistors are up-sized to give a better SNM, they consume more power and occupy a larger area. Hence, up-sizing the transistors in SRAM cells results in area and power overhead. For *full,up*, overall area and power increases by 11.2% and 5%, respectively. In *comp,up*, area and power overhead is compensated by reducing the width of one of the register bank. Effective savings in area and power are 28% and 33%, respectively.

Figure 5.12(b) shows the combined area and power savings for the technique *comp,up* along with the NP predictor compared to the baseline. We can see that the total power savings due to our proposed approach varies from 30.68% to 14.86% as the predictor size is increased from 512 to 8k. We save area in all configurations except for the 8k predictor.



(a) Savings without predictor overhead.



(b) Savings with predictor overhead.

Fig. 5.12: Area and power savings for various configurations of the proposed register file.

Configuration *comp,up* combined with 8k predictor has an area overhead of 19.77%.

5.5.3 Comparison with Supply Voltage Scaling

Next, we evaluate performance of our technique against the supply voltage scaling. We increase the supply voltage of the baseline by 10%, 20%, and 30% and compare the resultant SNM with *comp,up*. Figure 5.13, 5.14, and 5.15 present benefits of the supply voltage scaling at the 22nm, 32nm, and 45nm technology node, respectively. For the 22nm technology node, average improvement in SNM due to the 20% voltage scaling is 16% only. Similarly, the improvement in SNM due to voltage scaling is very small at the 32nm and 45nm technology nodes. In addition, this improvement in SNM comes at higher power consumption. Clearly, our technique works better than the supply voltage scaling.

5.5.4 Effect of Transistor Sizing

To study the impact of transistor sizing, we find the SNM improvement of register file for different sizing parameters. We increase the width of transistors by 10%, 20%, and 30%

and compare the resultant SNM with the *comp,up*. Figure 5.16, 5.17, and 5.18 present results for the SNM improvement for different levels of the transistor sizing at the 22nm, 32nm, and 45nm technology nodes, respectively. For the 22nm technology node, average improvement in SNM for the 20% transistor sizing is 102%, compared to 125% with our technique. Similarly, the SNM improvement due to the transistor sizing is smaller than the *comp,up* at 32nm and 45nm technology nodes.

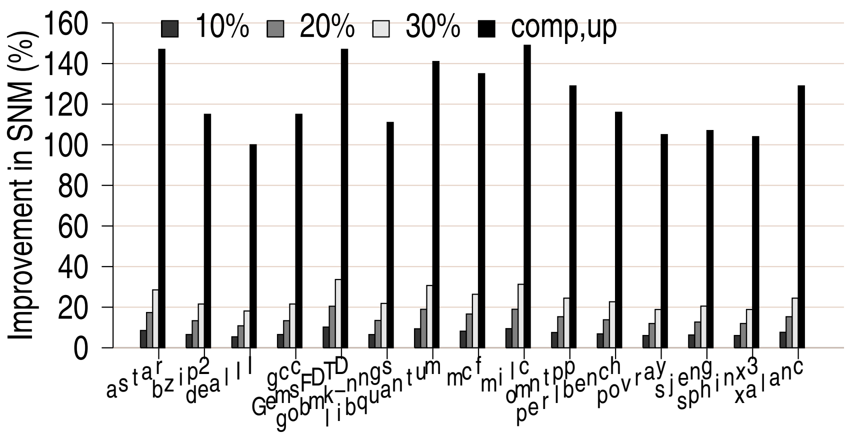


Fig. 5.13: Percentage improvement in SNM with different levels of supply voltage scaling at 22nm. The last bar in cluster shows our technique.

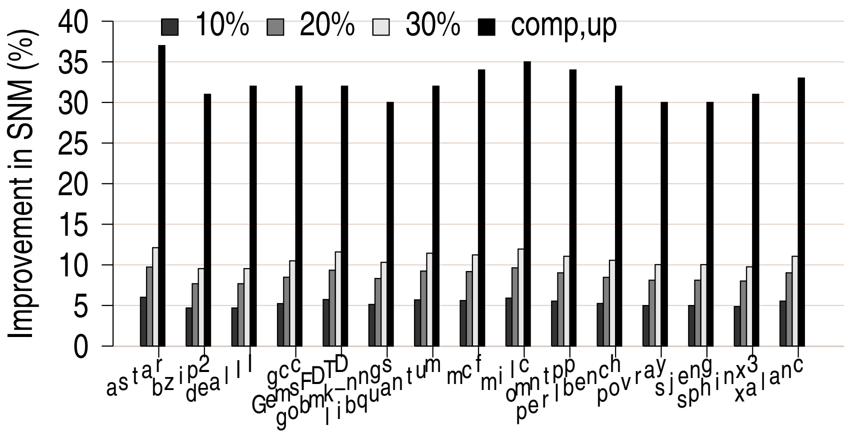


Fig. 5.14: Percentage improvement in SNM with different levels of supply voltage scaling at 32nm. The last bar in cluster shows our technique.

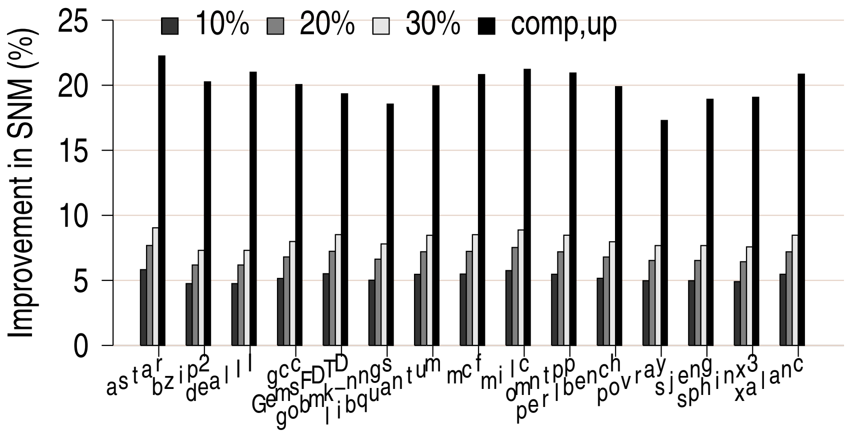


Fig. 5.15: Percentage improvement in SNM with different levels of supply voltage scaling at 45nm. The last bar in cluster shows our technique.

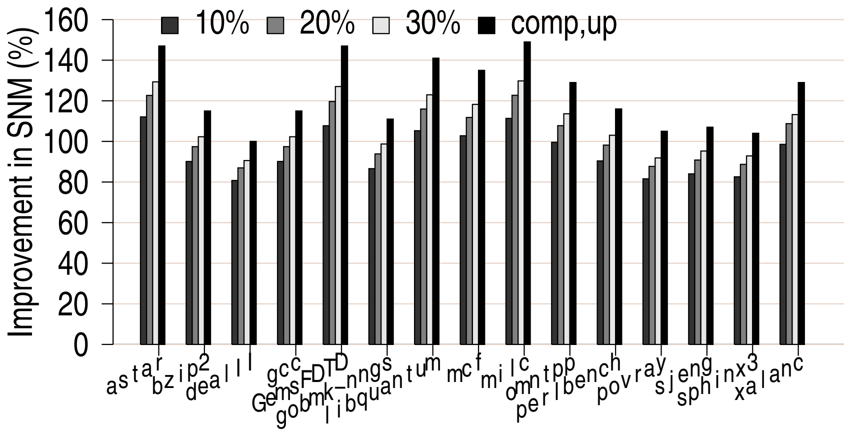


Fig. 5.16: Percentage improvement in SNM with different levels of transistor sizing at 22nm. The last bar in the cluster shows our technique.

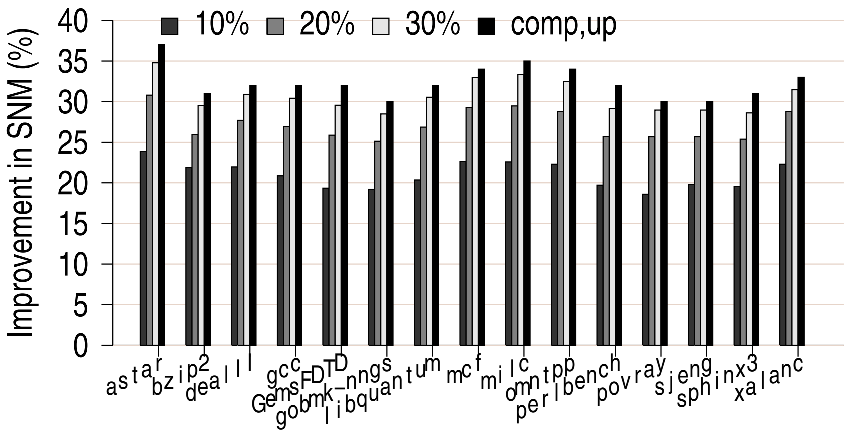


Fig. 5.17: Percentage improvement in SNM with different levels of transistor sizing at 32nm. The last bar in the cluster shows our technique.

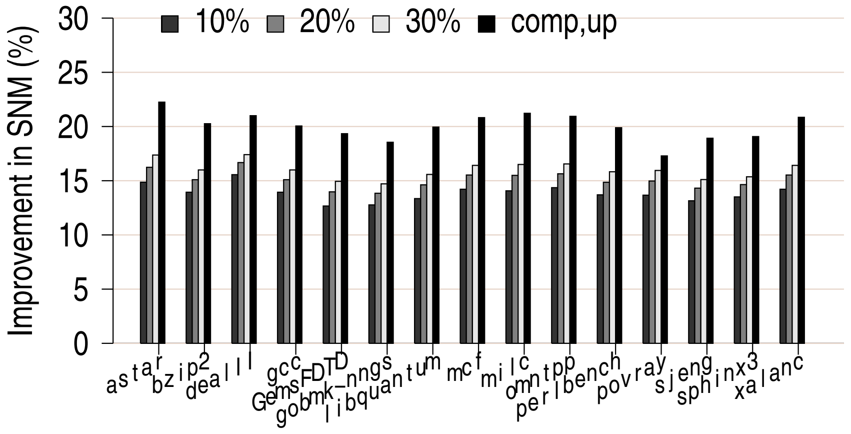


Fig. 5.18: Percentage improvement in SNM with different levels of transistor sizing at 45nm. The last bar in the cluster shows our technique.

Chapter 6
Conclusion

NBTI is one of the critical challenges for the semiconductor industry today. NBTI in the register file reduces overall reliability of the processor and potentially causes system failure. In this work, we proposed a novel approach for mitigating NBTI in both architecture and physical register file. For an architecture register file, we analyzed the reliability degradation when running different desktop applications, which can interleave in an arbitrary fashion. The key observation is that the application characteristics interleaving can lead to substantially severe degradation, in comparison to analysis with isolated applications. The micro-architecture techniques proposed in this paper are able to achieve a substantially robust design, leading to 2.2X improvement in SNM degradation, and 14X improvement in SNM uncertainty due to NBTI stress.

We mitigate the NBTI aging in a physical register file by looking at instructions producing large NBTI stress. We found that some of the instructions produce values that can cause high NBTI stress in the register file, while other instructions produce values which have limited impact. With this observation, we designed a prediction mechanism for detecting instructions that produce high NBTI stress. To increase overall reliability of the register file, we divided the register file into two banks and used up-sized transistors. We show that this approach provides 125% improvement in the average SNM of the register file. Our techniques also result in reduction of total area and power consumption.

References

[1] J. Abella, X. Vera, and A. González, “Penelope: The nbti-aware processor,” in *IEEE/ACM International Symposium on Microarchitecture*, pp. 85–96, 2007.

[2] T. Siddiqua and S. Gurumurthi, “Enhancing nbti recovery in sram arrays through recovery boosting,” *IEEE Transactions on VLSI Systems*, vol. PP, no. 99, p. 1, 2011.

[3] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, “Impact of nbti on sram read stability and design for reliability,” in *IEEE International Symposium on Quality Electronic Design (ISQED)*, pp. 210–218, 2006.

[4] G. Chen, K. Chuah, M. Li, D. Chan, C. Ang, J. Zheng, Y. Jin, and D. Kwong, “Dynamic nbti of pmos transistors and its impact on device lifetime,” in *IEEE International Reliability Physics Symposium Proceedings*, pp. 196–202, 2003.

[5] T. Grasser, B. Kaczer, P. Hehenberger, W. Gos, R. O’Connor, H. Reisinger, W. Gustin, and C. Schunder, “Simultaneous extraction of recoverable and permanent components contributing to bias-temperature instability,” in *IEEE International Electron Devices Meeting*, pp. 801–804, 2007.

[6] A. Islam, G. Gupta, S. Mahapatra, A. Krishnan, K. Ahmed, F. Nouri, A. Oates, and M. Alam, “Gate leakage vs. nbti in plasma nitrided oxides: characterization, physical principles, and optimization,” in *IEEE International Electron Devices Meeting*, pp. 1–4, 2006.

[7] M. Alam and S. Mahapatra, “A comprehensive model of pmos nbti degradation,” *Microelectronics Reliability*, vol. 45, no. 1, pp. 71 – 81, 2005.

[8] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula, “Scalable model for predicting the effect of negative bias temperature instability for reliable design,” *IET Circuits Devices and Systems*, vol. 2, no. 4, 2008.

[9] T.-B. Chan, J. Sartori, P. Gupta, and R. Kumar, “On the efficacy of nbti mitigation techniques,” in *Design Automation and Test in Europe (DATE)*, pp. 1–6, 2011.

[10] V. Reddy, A. Krishnan, A. Marshall, J. Rodriguez, S. Natarajan, T. Rost, and S. Krishnan, “Impact of negative bias temperature instability on digital circuit reliability,” in *40th Annual Reliability Physics Symposium Proceedings*, pp. 248–254, 2002.

[11] A. Bansal, R. Rao, J. Kim, S. Zafar, J. Stathis, and C. Chuang, “Impacts of NBTI and PBTI on SRAM static/dynamic noise margins and cell failure probability,” *Microelectronics Reliability*, vol. 49, no. 6, pp. 642–649, 2009.

[12] K. Kang, H. Kufluoglu, K. Roy, and M. A. Alam, “Impact of negative-bias temperature instability in nanoscale sram array: modeling and analysis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, pp. 1770–1781, 2007.

[13] H.-I. Yang, S.-C. Yang, W. Hwang, and C.-T. Chuang, “Impacts of nbti/pbti on timing control circuits and degradation tolerant design in nanoscale cmos sram,” *IEEE Transactions on Circuits and Systems*, pp. 1239 –1251, 2011.

[14] H. Yang, C. Chuang, and W. Hwang, “Impacts of NBTI and PBTI on power-gated SRAM with high-k metal-gate devices,” in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 377–380, 2007.

[15] T. Siddiqua and S. Gurumurthi, “A multi-level approach to reduce the impact of nbti on processor functional units,” in *Association for Computing Machinery (ACM) Great Lakes Symposium on VLSI*, pp. 67–72, 2010.

[16] A. Tiwari and J. Torrellas, “Facelift: Hiding and slowing down aging in multicores,” in *IEEE/ACM International Symposium on Microarchitecture*, pp. 129–140, 2008.

[17] Z. Qi and M. R. Stan, “Nbti resilient circuits using adaptive body biasing,” in *Association for Computing Machinery (ACM) Great Lakes Symposium on VLSI*, pp. 285–290, 2008.

[18] L. Li, Y. Zhang, J. Y. 0002, and J. Zhao, “Proactive nbti mitigation for busy functional units in out-of-order microprocessors,” in *Design Automation and Test in Europe (DATE)*, pp. 411–416, 2010.

[19] X. Fu, T. Li, and J. A. B. Fortes, “Nbti tolerant microarchitecture design in the presence of process variation,” in *IEEE/ACM International Symposium on Microarchitecture*, pp. 399–410, 2008.

[20] M. DeBole, R. Krishnan, V. Balakrishnan, W. Wang, H. Luo, Y. Wang, Y. Xie, Y. Cao, and N. Vijaykrishnan, “New-age: a negative bias temperature instability-estimation framework for microarchitectural components,” *International Journal of Parallel Programming*, vol. 37, pp. 417–431, 2009.

[21] Y. Wang, H. Luo, K. He, R. Luo, H. Yang, and Y. Xie, “Temperature-aware nbti modeling and the impact of input vector control on performance degradation,” in *Design Automation and Test in Europe (DATE)*, pp. 546–551, 2007.

[22] O. Khan and S. Kundu, “A self-adaptive system architecture to address transistor aging,” in *Design Automation and Test in Europe (DATE)*, pp. 81–86, 2009.

[23] W. Wang, S. Yang, S. Bhardwaj, S. Vrudhula, F. Liu, and Y. Cao, “The impact of nbti effect on combinational circuit: modeling, simulation, and analysis,” *IEEE Transactions on VLSI Systems*, vol. 18, no. 2, pp. 173–183, Feb. 2010.

[24] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula, “Predictive modeling of the nbti effect for reliable design,” in *IEEE Custom Integrated Circuits Conference*, pp. 189 –192, Sept. 2006.

[25] K. Agarwal and S. R. Nassif, “Statistical analysis of sram cell stability,” in *Design Automation Conference (DAC)*, pp. 57–62, 2006.

[26] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. River Edge, NJ: Addison-Wesley Publishing Company, 2010.

[27] W. Zhao and Y. Cao, “New generation of predictive technology model for sub-45nm early design exploration,” *IEEE Transactions on Electron Devices*, vol. 53, no. 11, pp. 2816 –2823, 2006.

[28] K. Kang, H. Kufluoglu, M. Alain, and K. Roy, “Efficient transistor-level sizing technique under temporal performance degradation due to nbti,” in *International Conference on Computer Design*, pp. 216 –221, Oct. 2006.

[29] X. Yang and K. Saluja, “Combating nbti degradation via gate sizing,” in *IEEE International Symposium on Quality Electronic Design (ISQED)*, pp. 47 –52, March 2007.

[30] X. Chen, Y. Wang, Y. Cao, Y. Ma, and H. Yang, “Variation-aware supply voltage assignment for minimizing circuit degradation and leakage,” in *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED ’09, pp. 39–44, 2009.

[31] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, “Adaptive techniques for overcoming performance degradation due to aging in digital circuits,” in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, pp. 284–289, 2009.

[32] *The SPARC Architecture Manual Version 9*, SPARC International, Inc, 1994.

[33] M. Lipasti, B. Mestan, and E. Gunadi, “Physical register inlining,” in *International Symposium on Computer Architecture*, pp. 325 – 335, June 2004.

[34] D. Brooks and M. Martonosi, “Dynamically exploiting narrow width operands to improve processor power and performance,” in *Proceedings. of High Performance Computer Architecture (HPCA)*, pp. 13–22, 1999.

[35] G. Loh, “Exploiting data-width locality to increase superscalar execution bandwidth,” in *IEEE/ACM International Symposium on Microarchitecture*, pp. 395 – 405, 2002.

[36] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, “Simics: a full system simulation platform,” *IEEE Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[37] K. Yeager, “The mips r10000 superscalar microprocessor,” *IEEE Micro*, vol. 16, no. 2, pp. 28–41, April 1996.

[38] T. Sherwood, E. Perelman, and B. Calder, “Basic block distribution analysis to find periodic behavior and simulation points in applications,” in *Parallel Architectures and Compilation Techniques*, pp. 3–14, 2001.

[39] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, “Cacti 6.0: a tool to model large caches,” *School of Computing, University of Utah, Technology Report*, 2007.