A Multi-Level Approach to Reduce the Impact of NBTI on Processor Functional Units

Taniya Siddiqua, Sudhanva Gurumurthi
Dept. of Computer Science
University of Virginia
{taniya,gurumurthi}@cs.virginia.edu

ABSTRACT

NBTI is one of the most important silicon reliability problems facing processor designers today. The impact of NBTI can be mitigated at both the circuit and microarchitecture levels. In this paper, we propose a multi-level optimization approach, combining techniques at the circuit and microarchitecture levels, for reducing the impact of NBTI on the functional units (FUs) of a high-performance processor core. We perform SPICE simulations to evaluate the impact of circuit-level design optimizations to reduce the NBTI guardband in terms of area, delay, and power. We then propose a set of NBTI-aware dynamic instruction scheduling policies at the microarchitecture level and quantify their impact on application performance and guardband reduction through execution-driven simulation. We show that carefully combining techniques at both these levels provides the most attractive solution to reducing the guardband while imposing the least overhead in terms of area, power, delay, and application performance.

Categories and Subject Descriptors

B.2.3 [**ARITHMETIC AND LOGIC STRUCTURES**]: Reliability, Testing, and Fault-Tolerance

General Terms

Reliability

1. INTRODUCTION

Silicon reliability is one of the key challenges facing the microprocessor industry today. There is a myriad of reliability issues that processor designers have to cope with, including soft errors, hard errors, and process variation. One of the key emerging hard error problems is Negative Bias Temperature Instability (NBTI). NBTI affects the lifetime of PMOS devices and occurs when a logic input of '0' (i.e., a negative bias) is applied at the gate of the transistor. This negative bias leads to the generation of interface traps at the Si/SiO_2 interface. These interface traps cause the threshold voltage of the transistor to increase, thereby lowering the speed of the device and the noise margin of the circuit which finally lead to a circuit failure [12, 9]. However, some of the interface traps can be eliminated by applying a logic input of '1' to the device, putting the device into a "recovery mode" [1].

NBTI affects both the cycle time and the stability of storage structures within the processor. These problems are typically addressed via guardbanding. Guardbanding accounts for the degradation in cycle time and the stability of the storage structures over the lifetime by reducing the operating frequency and increasing the minimum voltage of the storage elements (V_{min}). Typically, 20% of the cycle time is reserved as a guardband for NBTI and a 10% increase in threshold voltage (V_t) can be handled with a 10% increase in V_{min} [1]. However, reducing the frequency and increasing V_{min} have a detrimental impact on performance and power respectively and therefore it is desirable to reduce the guardband via the use of NBTI mitigation techniques. In this work, we look into guardband reduction techniques to address the degradation in cycle time.

Techniques for putting PMOS devices into the recovery mode can be implemented at both the circuit and microarchitecture levels in the processor. The goal of circuit-level techniques is to design the structures such that as many PMOS devices as possible in the structure can be put into the recovery mode whenever possible. Circuit-level techniques typically attempt to tackle NBTI at the granularity of a single structure. Microarchitecture level techniques, on the other hand, can manage NBTI for several structures within the processor core using techniques such as instruction fetch and scheduling policies. There are tradeoffs in implementing NBTI recovery at each of these levels in the system in terms of area, power, performance, complexity, and, their effectiveness in reducing the guardband.

In this paper, we present a quantitative analysis of NBTI recovery techniques at the circuit and microarchitecture levels for the functional units (FUs) in the cores of a high-performance multicore processor. We choose to study FUs because of the general trend in multicore processor design, where more cores are integrated onto the die each sucessive generation but the cores themselves tend to be relatively simple and have only a small number of FUs. In this scenario, the failure of even one FU could seriously jeopardize the ability of that core to provide high performance. We characterize the effectiveness of NBTI mitigation at both the circuit and microarchitecture levels and quantify their impact on other important figures of merit, such as, area, delay, and application performance. The objective of this chracterization study is to identify those techniques that can effectively reduce the guardband but have the least amount of performance impact on applications, as well as impose minimal overheads in terms of area, power, and delay. We then show that lightweight optimizations at each level is more effective in reducing the guardband without adversely affecting the other figures of merit than applying more extensive changes at any one level. To the best of our knowledge, this is the first paper to

GLSVLSI'10, May 16–18, 2010, Providence, Rhode Island, USA.

systematically analyze NBTI recovery techniques at each of these levels for FUs and develop multi-level optimization techniques to tackle this important reliability problem.

The specific contributions of this paper are:

- We propose and evaluate three different NBTI-aware FU designs in terms of guardband reduction, area and delay. We show that, on an average, the three different designs provide a guardband reduction of 42%, 46% and 47% over the baseline configuration.
- We then propose three different NBTI-aware scheduling policies and evaluate their impact on performance and guardband reduction. We show that, on average, the three policies provide a guardband reduction of 43%, 54% and 63% over the baseline FU configuration.
- Finally, we analyze the effectiveness of combining lightweight optimizations at each level and show that this approach provides a 56% guardband reduction with minimal impact on other figures of merit.

The outline of the rest of this paper is as follows. The next section discusses the related work and the NBTI mitigation techniques we consider are described in Section 3. The experimental methodology is described in Section 4. The results are presented in Section 5 and Section 6 concludes this paper.

2. RELATED WORK

NBTI mitigation techniques roughly fall into two categories: those that aim to reduce the stress on the PMOS devices by controlling V_{dd}, V_t, and temperature, and those that try to increase the recovery time to facilitate detrapping.

There have been several studies into enhancing the reliability of FUs at both the circuit and microarchitecture levels. [1, 6] tackle NBTI-induced wearout problems by feeding specific inputs into the FUs during idle periods to increase the recovery time. [16] uses temperature-based job-scheduling to individual cores of a multi-core processor in conjunction with V_{dd} and V_t control to hide the effects of aging due to NBTI. There also has been prior work on fault tolerant designs and hard error detection schemes for FUs. [3] proposes a microprocessor design integrated with a dynamic verification based error detection mechanism to diagnose hard faults in FUs. [7] presents a fault-tolerant design of the Kogge-Stone Adder that leverages its inherent redundancy to continue reliable operation even in the presence of hard errors.

While all these prior works propose optimizations at the circuit or microarchitecture level, to our knowledge there is no systematic analysis of NBTI optimizations for the FU at each level and an exploration of multi-level optimization techniques to tackle this problem. Fu et al. [5] propose multi-level optimization for soft error mitigation for SRAM arrays in SMT processor cores and show that such combined approaches are attractive in terms of reliability, performance, and power. In this paper, we show that a multi-level approach is more attractive for NBTI mitigation in the FUs of a high-performance processor than a circuit-only or microarchitecture-only approach.

3. APPROACHES TO NBTI MITGATION AT THE CIRCUIT AND MICROARCHITECTURE LEVELS

In this section, we describe how degradation due to NBTI can be reduced at the circuit and microarchitecture levels for the functional units (FUs) and describe the specific designs and policies we evaluate. While our circuit designs are optimized versions of a previously proposed technique [6], the microarchitecture level schemes we propose are novel.

3.1 Circuit Level Techniques

Fu et al. [6] propose a circuit-level technique where a FU is divided into multiple segments and instructions with narrow-width operands are steered into one of the segments based on the delay of each segment due to NBTI. The delay of a segment depends on the level of stress experienced by the PMOS transistors in that segment. Higher the stress, more would be the increase in V_t and hence higher would be its delay. The specific design considered in [6] is a 64-bit FU that is divided into four segments of 16 bits each. An instruction with 16-bit or smaller operands uses the segment with the smallest delay while the other segments are put into the recovery mode. Instructions with operand-widths greater than 16 bits make use of the entire FU.

There are two drawbacks to the design proposed by Fu et al. First, each segment in the FU is built as a Carry Lookahead Adder (CLA) and the segments are connected as a multi-level CLA to form a 64-bit FU. CLAs are seldom used in high performance microprocessors due to their low speed. Instead, most processors today use some form of a high-speed prefix adder. Second, to put PMOS devices in the idle segments into the recovery mode, they feed in special input vectors. Since an FU is a complex combinational circuit, a single vector of bits input to the FU cannot put all the PMOS devices in it into the recovery mode and instead a sequence of input vectors are necessary, which takes multiple clock cycles. Therefore, the PMOS devices cannot utilize the entire duration of an idle period to recover from NBTI. We address both these limitations as follows.

In this paper, we model the FU to be a 64-bit Kogge-Stone adder (KSA) which is a high-speed prefix adder [8] for the purpose of estimating NBTI. KSA is a widely used FU design due to its regular structure and its speed. In order to put all the PMOS devices in the FU into the recovery mode during an idle period, we make use of power gating [10]. Power gating reduces leakage power in the circuit by using a header or footer device which connects the entire circuit to V_{dd} or ground. Whenever the circuit is idle, turning off this transistor disconnects the circuit from V_{dd} or ground. In our model, we use footer devices to connect the circuit to ground. During power gating, we find that the gate voltages of the PMOS devices in the FU stay reasonably close to a logic value of '1', thereby putting the FU into the recovery mode.

We design the KSA such that it consists of segments that are capable of processing narrow-width operands while the idle segments undergo recovery using power gating. We profiled the SPEC CPU2000 benchmark suite [15] to determine the distribution of narrow-width operands. We find that there are a large number of instructions with 8-bit, 16-bit, and 32-bit operands. Therefore, there is significant scope for NBTI recovery by partitioning the 64-bit FU into 2, 4, or 8 segments, where each segment is 32 bits, 16 bits, or 8 bits respectively. Each such segment can operate as an independent FU on operands of the given width and multiple such segments can be combined to operate on wider operands. For example, if the FU is partitioned into 4 parts, an instruction with two 16-bit input operands needs to utilize only one part while the other three can be put into the recovery mode. An instruction with 32-bit operands can use either the first two or last two consecutive segments while the other two are put into the recovery mode. Figure 1 shows an FU that has two segments. To ensure that all parts of the FU experience roughly an equal amount of wear, we schedule instructions to the segments in a round-robin fashion.

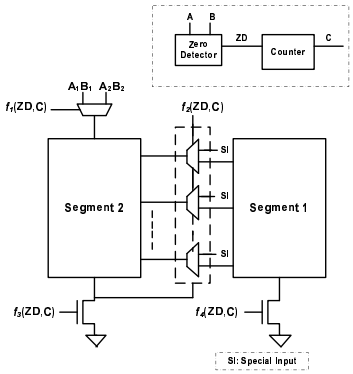


Figure 1: Partitioned Kogge-Stone Adder Design to Support NBTI Recovery.

It is important to note that the KSA is fast because the carries are computed in parallel, a feature that we retain in the partitioned design. The delay of the FU, when operating in the 64-bit mode, should not be significantly affected by the partitioned nature of the design. In order to achieve this property, we introduce a set of MUXes between adjacent segments, as shown in Figure 1. The selection input to these MUXes depend on the width at which the FU will be used. For example, in the case of Figure 1, the input to the MUXes will determine whether the FU will be used as a 64-bit FU or a 32-bit FU. Each segment is connected to ground via a footer device. The gates of the footer devices are controlled by zero detectors and a one-bit counter output. The purpose of the zero detectors is to decide whether the instruction has narrow-width operands or not. The one-bit counter is incremented every time a narrow-width operation is performed. If the operands are narrow-width, they are steered towards a particular segment using a round-robin policy based on the value of the counter. The footer for the left segment is shared with the MUXes since the MUXes provide the input to the left segment.

There are tradeoffs in designing the FU in a partitioned manner to support recovery. The more segments that a FU has, greater are the opportunities for recovering the PMOS devices and reducing the guardband, since there will be a better matching between narrow-width operands and the number of segments required to operate on them. However, such a design comes at the cost of increased area, delay, and higher power consumption when it is used for operating on wide operands. We analyze these tradeoffs in Section 5.

3.2 Microarchitecture Level Techniques

There are opportunities to reduce the guardband by carefully managing the hardware resources within the core at the microarchitecture level. The usage characteristics of different FUs can be controlled through various instruction scheduling policies. The dynamic instruction scheduler decides which instructions are executed and at what times on a given set of FUs and therefore has a strong impact on the utilization characteristics of the FU. The scheduler consists of two key components: wakeup logic and select logic. The wakeup logic is responsible for asserting an instruction as being 'ready' in the issue window by updating the source dependences of instructions waiting for their source operands to become available. Every time a result is produced by a FU, the tag of the result is broadcast to the waiting instructions in the issue window. Each waiting instruction compares the result tag with the tag of each of its source operands. Once both operand tags have matched, the instruction is ready to execute (instruction wakeup) and the ready instructions signal the select logic to request execution on a given type of FU. Once a FU becomes available, the select logic directs a suitable instruction to that unit for execution by asserting the corresponding grant signal (instruction select). Since multiple instructions could wakeup in a given cycle and the processor typically has multiple FUs of the same type, there needs to be a policy to select a subset of the ready instructions and assign them to specific FUs based on resource availability. Modern instruction schedulers typically use a form of prioritized scheduling where instructions are selected in an oldest-first order from the issue window and each instruction is issued to the lowest-numbered FU that is free. We call this policy **Prioritized Scheduling** (PS). In this approach, an instruction will be allocated to FU_0 if it is available; if FU_0 is busy, then FU_1 will be checked for availability and so on. This non-uniform assignment leads to the case where FUs with smaller sequence numbers get utilized more than those with higher sequence numbers and hence degrade faster. This is illustrated in Figure 2, which presents the utilization of the integer ALUs (in terms of the number of cycles that the FU is busy over the entire execution time of the workload) of a 4-wide issue processor core for a set of SPEC CPU2000 benchmarks. As we can observe from the figure, the lower-numbered functional units tend to be heavily utilized and therefore will wear out sooner than the higher-numbered ones.

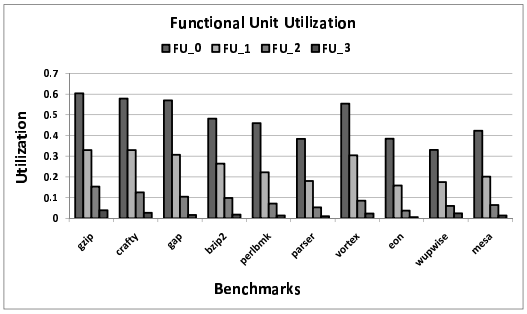


Figure 2: Utilization of the Integer ALUs in a 4-wide issue processor core using the PS policy.

We propose instruction scheduling policies that attempt to extend the recovery times for the FUs so that their degradation due to NBTI can be minimized. We make use of power gating to put idle FUs into the recovery mode. We now present a brief overview of the instruction scheduling policies that we evaluate.

i) Priority Rotation Scheduling (PR): The PR policy is geared towards achieving a balanced utilization of the FUs in order to level the wear on them. This policy modifies the conventional PS scheduling policy so that the priorities of the FUs are changed, in a round-robin fashion, after a fixed number of cycles ($\texttt{Cycle}_{PR}$) have elapsed. In the PR policy, we start with FU_0 having the highest priority and assign lower priorities to the other FUs based on their sequence numbers (i.e., FU_0 initially has the highest priority whereas FU_{n-1} has the lowest). After $\texttt{Cycle}_{PR}$ cycles, FU_1 gets the highest priority, FU_2 gets the second highest priority and so on and FU_0 has the lowest priority. The select logic is similar to the PS policy but with added functionality to change the priorities of the FUs after $\texttt{Cycle}_{PR}$ cycles. A key advantage of the PR policy is that it does not degrade performance in the sense that no FU is precluded from being used because it is in the recovery mode.

ii) Time-Dependent Scheduling (TD): Time-Dependent Scheduling extends the conventional policy to include an explicit fixed recovery period. In the TD policy, whenever the wakeup logic flags an instruction to be ready, the select logic allocates the lowest numbered FU that is available. After a FU is used, no other instruc-

tion is assigned to that particular FU for a fixed number of cycles ($\texttt{Cycle}_{TD}$). This policy allows the FU to undergo recovery for $\texttt{Cycle}_{TD}$ cycles after each stress phase. We can implement this in hardware by keeping the busy signal of the FU asserted for $\texttt{Cycle}_{TD}$ cycles after the FU is used. However, since the FU cannot be used during this time, there could be a detrimental impact on performance.

iii) Prioritized Time-Dependent Scheduling (PTD): Prioritized Time-Dependent Scheduling combines the PR and TD policies to include an explicit fixed recovery period to the highest priority FU. In this scheme, similar to PR policy, the priorities of the FUs are changed in a round-robin fashion after a fixed number of cycles ($\texttt{Cycle}_{PRC}$) have elapsed. Whenever an FU gains the highest priority, it also gains the priviledge of an extended recovery period after it is used. When this high priority FU is used, no other instruction is assigned to this particular FU for another fixed number of cycles ($\texttt{Cycle}_{TDC}$), allowing it to undergo recovery after each stress phase. Other FUs can be continued to be used as usual. Once $\texttt{Cycle}_{PRC}$ cycles have elapsed, the priorities are rotated. This transition in priorities does not affect the residual time for which an FU can remain in the recovery mode. Any FU that enters the recovery mode is guaranteed to be in that mode for $\texttt{Cycle}_{TDC}$ cycles. Since the PTD policy prevents an FU that is in the recovery mode from being used till the requisite number of cycles elapse, this policy could also have a detrimental impact on performance. However, if $\texttt{Cycle}_{PRC}$ is chosen to be significantly larger than $\texttt{Cycle}_{TDC}$, only one FU would tend to be in the recovery mode at any one time and therefore the degradation in performance would tend to be less severe than the TD policy. We evaluate these policies in Section 5.

4. EXPERIMENTAL SETUP

Our circuit-level modeling is performed via SPICE-level simulation using the Cadence Virtuoso Spectre circuit simulator [4] for the 32nm process using the Predictive Technology Model [11]. Our architecture-level evaluations are carried out via execution-driven simulation using the M5 simulator [2]. We simulate a 4-wide issue core, which is representative of cores used in multicore processors today, that runs at a 3 GHz clock frequency and has a supply voltage of 0.9V. We use all 26 benchmarks from the SPEC CPU2000 benchmark suite in our evaluations [15]. 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 [13].

In this paper, we focus on the impact of NBTI on the integer ALUs only. Although we consider both integer and floating-point benchmarks in our evaluations, several of the floating-point benchmarks have a considerable number of integer instructions in their instruction mix and therefore make heavy use of the the integer ALUs [14]. We present our results for only the integer ALU with the lowest sequence number since this ALU tends to be the most heavily utilized of all the ALUs with conventional instruction scheduling, as explained in Section 3.2. Since NBTI affects the threshold voltages of PMOS devices in the FUs, the delay of the FU hardware increases which causes a potential danger to meet the timing constraints. Guardbanding is used to protect a circuit from failure, which entails both performance and power penalties. In all our experiments, we assume that the baseline processor uses the unpartitioned FU design, the PS instruction scheduling policy, and a guardband of 20% [1].

In our evaluations, we use the percentage reduction in the guardband with respect to the baseline as the figure of merit to quantify the extent to which a particular design mitigates the impact of NBTI. To compute guardband reduction, we track the stress and recovery cycles of the FUs in the architectural simulations. Using these statistics, we estimate the degradation in V_t after a 7-year service life [16], using which we calculate the delay degradation in the structures and finally the guardband reduction. We also quantify the impact on other important figures of merit, such as, area, delay, and application performance (in terms of IPC), to study the effectiveness of NBTI mitigation at both the circuit and microarchitecture levels.

Our goal is to evaluate the extent to which optimizations at only the circuit or microarchitecture level can reduce the guardband and ascertain the costs incurred in applying these optimizations on the other figures of merit. These results are presented in Sections 5.1 and 5.2. Based on this analysis, in Section 5.3, we study whether less aggressive optimizations at each level, which impose less overheads, can be combined to provide an effective means to reduce the guardband.

5. RESULTS

5.1 Circuit-Level Optimization

We now study the tradeoffs between different circuit-level techniques. We partition the FU into 2, 4 and 8 segments and analyze the guardband reductions, area overheads, and the increases in delay with respect to the baseline.

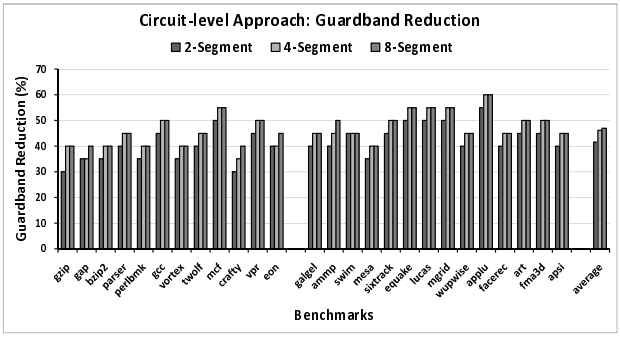
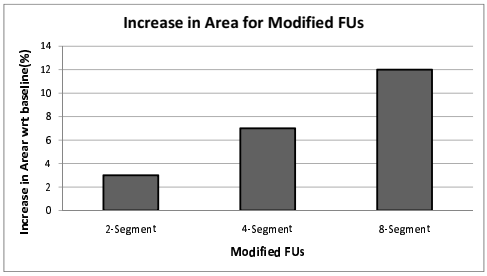


Figure 3: Guardband reduction for the FUs with 2, 4 and 8 segments.

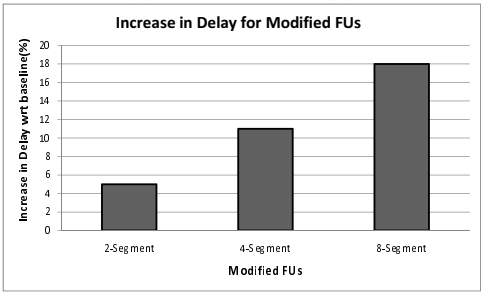
Guardband Reduction: Figure 3 presents the guardband reduction (as a percentage) due to the modified FU designs. The FU with 2 segments provides a guardband reduction of 30%-55% whereas a 4-Segment FU and a 8-Segment FU provide 35%-60% and 40%-60% guardband reductions respectively. This result shows that the more partitions that a FU has, the higher would be the guardband reduction. However, we see that the guardband reduction achieved by going from the baseline to the 2-segment design is much higher than going from the 2- to 4-segments and 4- to 8-segments. Also, the integer benchmarks, which are the left-hand side group of bars in the figure 3, experience less guardband reductions than the floating-point benchmarks.

In order to understand how the modified FU design impacts NBTI, we need to analyze how the FUs are utilized and the distributions of the narrow-width operands. The instruction mix gives an indication of how frequently each type of FU gets accessed. For example, the higher the number of integer instructions, the higher is the probability of accessing integer FUs. A previous study by Siddiqua et al. [14] gives the breakdown of the instruction mix of these benchmarks. We find that the lowest guardband reductions are observed for those benchmarks which have a high percentage of integer instructions. Similarly, to understand the benefit of the

segmented designs, it is important to look at the distribution of the narrow-width operands. We find that, on average, the percentage of 32-bit operands is about 48% whereas it is 8% for 16-bit operands and 27% for 8-bit operands across the benchmark suite. Since the 32-bit operand sizes occur most frequently, we get a higher guardband reduction when we go in for the 2-segment partition from the baseline, whereas the 4-segment to 8-segment designs provide diminishing returns.



(a) Increase in Area (%)



(b) Increase in Delay (%)

Figure 4: Percentage increase in area and delay for the FUs with 2, 4 and 8 segments wrt. the unpartitioned FU design.

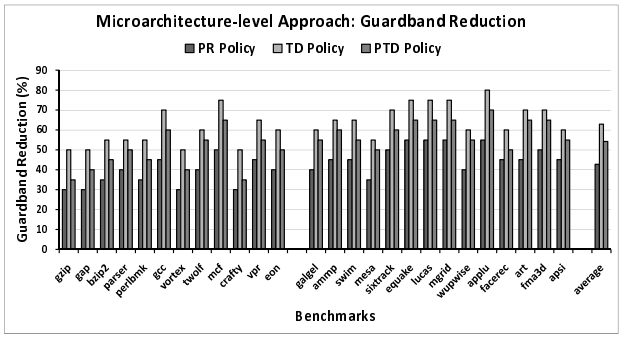
Area and Power: We design the FU for the baseline design to occupy the minimum area required to provide correct functionality. The overheads for the partitioned FU designs are given in Figure 4 (a). We can see that the area overhead of the 2-segment, 4-segment and 8-segment FUs are 3%, 7% and 12% respectively. The increase in area is due to the number of sets of MUXes required for the three designs (1, 3 and 7 respectively). These MUXes increase the power consumption of the FU and therefore the designs with more segments will consume more power.

Delay: We evaluate the delay to measure the highest clock frequency at which the FU can operate reliably. Figure 4 (b) shows the increase in delay with respect to the baseline design. In our simulations, we find that the increase in delay for the 2-segment, 4-segment and 8-segment FU designs due to the additional hardwares are 5%, 11% and 18% respectively. However, these increases in delay can be accomodated within a 333ps clock cycle time (which corresponds to the 3 GHz clock frequency) and therefore all these FU designs can provide a single-cycle access latency for instructions using the FUs.

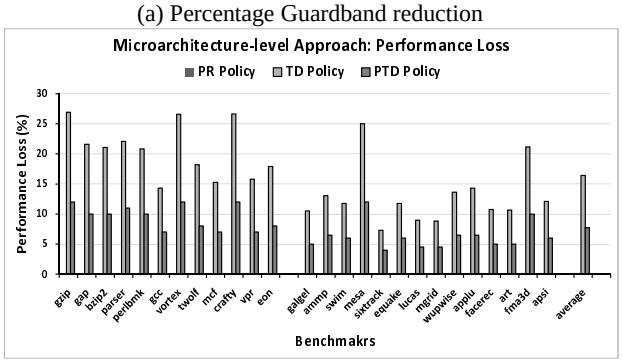
Summary: There are merits and demerits to using the aforementioned FU designs. The higher the segment count in the FU, the higher is the guardband reduction. However, the area, power, and delay overheads also increase significantly. Each segment introduces a set of extra circuitry which adds area and increases power consumption. For 64-bit operand computations, the FU would consume the highest power since all the MUXes will be active and this power consumption will be higher with more partitions. Overall, we find that we get higher guardband reduction with least area, power and delay overheads for the 2-segment design. Therefore, we choose the 2-segment FU design for use in the multi-level approach in Section 5.3

5.2 Microarchitecture-Level Optimization

We evaluate the guardband reduction and performance of the PR, TD and PTD policies. We use a value of 10K for $\texttt{Cycle}_{PR}$ and $\texttt{Cycle}_{PRC}$ and a value of 1 for $\texttt{Cycle}_{TD}$ and $\texttt{Cycle}_{TDC}$ for the different policies.



(a) Percentage Guardband reduction



(b) Percentage loss in performance

Figure 5: Impact of the instruction scheduling policies.

Guardband Reduction and Performance: Figure 5(a) presents the percentage guardband reduction for the various policies. The PR policy reduces the guardband by 30%-55% whereas the TD and PTD policies reduce it by 50%-80% and 35%-70% respectively. Since the scheduling policies aim to achieve improved guardband reduction by increasing idleness to provide NBTI recovery, they may impact performance, as discussed in Section 3.2. Figure 5(b) quantifies the performance loss. As we can see, the TD and PTD policies achieve a greater guardband reduction at the cost of reduced performance whereas the PR policy experiences no performance loss. On average, the TD and PTD policies lead to a performance loss of 17% and 8% respectively. The PTD policy provides a guardband reduction and experiences a performance loss that is between PR and TD because the value of $\texttt{Cycle}_{PRC}$ is much higher than $\texttt{Cycle}_{TDC}$. Therefore, the policies that attempt to increase the recovery time of a FU cause a corresponding loss in performance for the benchmarks. Similar to the circuit-level results, the integer benchmarks have less guardband reduction than the floating-point benchmarks.

The impact of scheduling policies on NBTI is influenced by two factors: the instruction mix and instruction level parallelism. As mentioned before, the instruction mix is an indicator of how frequently each type of FU gets accessed. Instruction level parallelism also determines the frequency of use of the FUs. Also, it affects how groups of FUs get used. When the IPC is high, more instructions are executed per cycle. Consequently, more FUs will get uti-

lized. Indeed we find the least guardband reductions are observed for those benchmarks which have a high percentage of integer instructions and higher IPC.

Summary: As the results indicate, there is a clear tradeoff between achieving higher guardband reductions and application performance by using instruction scheduling. Although the TD policy provides large benefits in terms of guardband reduction, it imposes large performance overheads as well, which is unattractive for use in high-performance processors. Although the PTD policy reduces the performance overheads significantly while still providing significant guardband reductions, the detrimental impact on application performance is still non-trivial. The PR policy, on the other hand, does not cause any performance loss while still providing a guardband reduction of 43% on average. Therefore, although TD and PTD provide large guardband reductions, we choose PR, since it is the most lightweight policy in terms of its performance impact, and use it in the multi-level approach.

5.3 Multi-Level Optimization

In the results presented thus far, we have observed that increasing the effectiveness of NBTI mitigation techniques at only the circuit or microarchitecture level to improve the guardband entails penalties in terms of area, power, delay, and application performance. On the other hand, lightweight optimizations at *each* level could provide a net increase in guardband reduction with less overhead. We now evaluate this hypothesis.

We choose the 2-segment FU design, which we found to have the smallest overhead in terms of area, delay, and power in Section 5.1 as our circuit-level optimization and the PR instruction scheduling policy as our microarchitecture-level optimization. The result of combining these two optimizations is given in Figure 6.

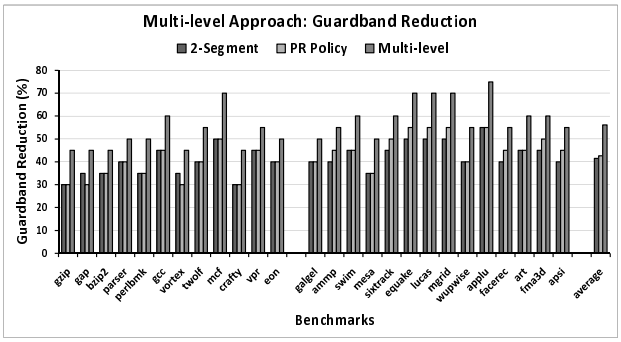


Figure 6: Guardband reduction using FUs with 2 segments and the PR scheduling policy.

From the figure, it is evident that we achieve a guardband reduction of 45%-75%, which is a much higher range than the 30%-55% for both circuit and microarchitecture levels. It is also important to note that the achieved guardband reduction due to the multi-level approach is *not* merely additive from the individual optimizations at each level. This is due to the fact that, in the multi-level approach, the overall flow of bits through the FUs over the course of execution of the workload is different from the previous two sets of evaluations. The new FU design changes the stress and recovery characteristics on the PMOS devices in the FU, due to the partitioned nature of its design, compared to the unpartitioned FU design evaluated in Section 5.2 for the microarchitecture-only optimizations. Similarly, the PR policy alters the overall utilization of each FU and increases the idleness of all the FU segments, which allows for greater recovery than the PS policy used in Section 5.1 for the circuit-only optimizations.

Overall, we observe that the multi-level approach provides greater reductions in the NBTI guardband while retaining the low area, power, and delay benefits of the 2-segment FU design and the high performance of the PR policy.

6. CONCLUSION

NBTI is one of the most important silicon reliability problems facing processor designers. In this paper, we evaluate both circuit and microarchitecture level approaches to reduce the NBTI guardband for the FUs of a high-performance processor core. At the circuit-level, we use an optimized version of a partitioned FU design and evaluate several design points in terms of their effectiveness in reducing the guardband and also their area, delay, and power. At the microarchitecture-level, we propose and evaluate a set of NBTI-aware dynamic instruction scheduling policies and evaluate their impact in terms of guardband reduction and performance. Finally, we show that a multi-level optimization approach, which combines the benefits of both circuit and microarchitecture level optimizations, is the most effective in reducing the guardband while imposing little overhead in terms of area, power, delay, and performance.

7. ACKNOWLEDGMENTS

We thank Mircea Stan for his valuable feedback. This research has been supported in part by NSF grant 0627527 and gifts from Intel.

8. REFERENCES

[1] J. Abella, X. Vera, and A. Gonzalez. Penelope: The NBTI-Aware Processor. In *Proceedings of the 40th IEEE/ACM International Symposium on Microarchitecture*, 2007.

[2] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, 26(4):52–60, July 2006.

[3] F. Bower, D. Sorin, and D. Ozev. A Mechanism for Online Diagnosis of Hard Faults in Microprocessors. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, November 2005.

[4] Cadence Virtuoso Spectre Circuit Simulator. http://www.cadence.com/products/cic/spectre_circuit/.

[5] X. Fu, T. Li, and J. Fortes. Combined Circuit and Microarchitecture Techniques for Effective Soft Error Robustness in SMT Processors. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, June 2008.

[6] X. Fu, T. Li, and J. Fortes. NBTI Tolerant Microarchitecture Design in the Presence of Process Variation. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, November 2008.

[7] S. Ghosh, P. Ndai, and K. Roy. A novel low overhead fault tolerant kogge-stone adder using adaptive clocking. In *Proceedings of the conference on Design, automation and test in Europe*, 2008.

[8] P. Kogge and H. Stone. A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations. *IEEE Transactions on Computers*, C-22(8):786–793, August 1973.

[9] S. Kumar, C. Kim, and S. Sapatnekar. Impact of NBTI on SRAM Read Stability and Design for Reliability. In *Proceedings of the International Symposium on Quality Electronic Design*, 2006.

[10] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. Vijaykumar. Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories. In *Proceedings of the International Symposium on Lower Power Electronics and Design (ISLPED)*, pages 90–95, July 2000.

[11] Predictive Technology Model. http://www.eas.asu.edu/˜ptm/.

[12] D. Schroder and J. Babcock. Negative Bias Temperature Instability: Road to Cross in Deep Submicron Silicon Semiconductor Manufacturing. *Journal of Applied Physics*, 94(1), July 2003.

[13] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, October 2002.

[14] T. Siddiqua and S. Gurumurthi. Balancing soft error coverage with lifetime reliability in redundantly multithreaded processors. In *IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, 2009.

[15] SPEC CPU2000. http://www.spec.org/cpu2000/.

[16] A. Tiwari and J. Torrellas. Facelift: Hiding and Slowing Down Aging in Multicores. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, November 2008.