Balancing Soft Error Coverage with Lifetime Reliability in Redundantly Multithreaded Processors

A Thesis

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Master of Science

Computer Science

by

Taniya Siddiqua

September 2009

Approvals

This dissertation is submitted in partial fulfillment of the requirements for the degree of

Master of Science

Computer Science

Taniya Siddiqua

Approved:

Sudhanva Gurumurthi (Advisor)

Kevin Skadron (Chair)

Mircea R. Stan

Accepted by the School of Engineering and Applied Science:

James H. Aylor (Dean)

September 2009

Abstract

Silicon reliability is a key challenge facing the microprocessor industry. Processors need to be designed such that they are resilient against both soft errors and lifetime reliability phenomena. However, techniques developed to address one class of reliability problems may impact other aspects of silicon reliability. In this thesis, we show that Redundant Multi-Threading (RMT), which provides soft error protection, exacerbates lifetime reliability. We then explore two different architectural approaches to tackle this problem, namely, Dynamic Voltage Scaling (DVS) and partial RMT. We show that each approach has certain strengths and weaknesses with respect to performance, soft error coverage, and lifetime reliability. We then propose and evaluate a hybrid approach that combines DVS and partial RMT. We show that this approach provides better improvement in lifetime reliability than DVS or partial RMT alone, buys back a significant amount of performance that is lost due to DVS, and provides nearly complete soft error coverage.

Acknowledgements

I am extraordinarily fortunate to work with my advisor Professor Sudhanva Gurumurthi for his sage guidance and I cannot thank him enough. He has always been a continuous source of inspiration and encouragement. No matter what, he has always been accessible and willing to help with research making it smooth and rewarding. I have learnt a great deal from him and undoubtedly will continue to learn. Inspite of my uncountable mistakes, he always encourages me to do the right thing with patience. I cannot hope to enumerate all his support throughout these two years. He has been a wonderful mentor. His intellect, patience and mentorship to both my academic and personal growth makes him a pleasure to work with.

I would like to thank all my best friends who stand by me in my tough moments, help me to recollect my enthusiasm about life and work. I would like to thank my friends at University of Virginia without whom life could have been very difficult. All of them have always been a source of support and fun.

Finally, my family without whom I cannot think of my existence. At times I lose all my hopes but my parents never let me feel down. They make me realize what is life about and I should always keep going. I am so much lucky to have all my caring siblings and adorable nephews/nieces. Lastly, I have no words to express gratitude and respect for my wonderful brother-in-law who is more than a brother to me. Without his moral support I would not be here.

Last but not least, thanks to God for making my life bountiful and giving me the strength to survive the tests.

Contents

Abstract iv

Acknowledgements v

List of Tables vii

List of Figures viii

1 Introduction **1**

2 Background and Related Work **4**

2.1 Soft Errors 4

2.2 Lifetime Reliability 5

3 Experimental Setup **8**

4 Results **12**

4.1 Impact of Redundant Multi-Threading on Processor Lifetime Reliability 13

4.2 Improving Lifetime Reliability through Dynamic Voltage Scaling . . . 16

4.3 Improving Lifetime Reliability through Partial Redundant Multi-Threading 21

4.4 Combining DVS with Partial RMT 27

5 Conclusions and Future Work **33**

List of Tables

3.1 Area of each structure as a percentage of the area of the structures within the Sphere of Replication 10

4.1 Baseline processor configuration 13

4.2 Percentage of integer and floating-point instructions in the benchmarks. 17

4.3 Architectural Vulnerability Factors of key structures within the Sphere of Replication in the single-threaded mode and for the $PRMT_{83}$, $PRMT_{85}$, and $PRMT_{87}$ partial RMT policies. 23

4.4 Architectural Vulnerability Factors of key structures within the Sphere of Replication for the HYB_{85} policy. The single-threaded mode and $PRMT_{85}$ AVFs are shown to facilitate data comparison. 30

List of Figures

3.1 SRT Processor Floorplan 9

4.1 Degradation in processor lifetime reliability due to SRT. 14

4.2 Impact of Dynamic Voltage Scaling on lifetime reliability and performance. 18

4.3 Impact of partial RMT on lifetime reliability and performance. . . . 25

4.4 Impact of the hybrid partial RMT scheme for the 85 C temperature threshold. 29

Chapter 1

Introduction

Silicon reliability is a key challenge facing the microprocessor industry today. Lower supply voltages and increased transistor integration densities are making processors increasingly vulnerable to soft errors due to external high-energy particles such as neutrons. Particle-induced soft errors are a problem for both memory arrays as well as the latches and logic within the processor core [24]. In addition, continued scaling has also increased the susceptibility of processors to several lifetime reliability phenomena, such as Electromigration, Time-Dependent Dielectric Breakdown, and Thermal Cycling [29, 30]. Therefore, processors have to be designed to provide adequate protection against both soft errors and lifetime reliability.

A number of architecture-level solutions have been proposed to tackle silicon reliability problems. Soft error mitigation techniques include Redundant Multithreading (RMT) [19, 13, 27] and Architecture Vulnerability Factor (AVF) reduction techniques [34]. There have also been a number of "partial" RMT techniques proposed in recent years, wherein the some amount of soft error coverage is sacrificed to improve performance or reduce power consumption [15, 11, 16, 17]. In the area of lifetime reliability, the proposed protection techniques include Dynamic Reliability Management [29], spatial redundancy [31], and dynamic verification [5, 7]. Previous works have focused

on addressing soft errors or lifetime reliability individually and have not considered how techniques that are developed for one class of reliability problems may impact other aspects of silicon reliability. However, in order to design reliable processors, it is important to consider these interactions. For example, the redundant threads of a single program in RMT could lead to increased activity within the chip, which could in turn accelerate the wear out of certain on-chip structures. Although there have been studies into reducing the performance overheads of RMT [15, 11, 16, 17], to the best of our knowledge there has been no prior work that has explored how soft error protection mechanisms impact lifetime reliability. Since soft errors and lifetime reliability are both important problems, techniques that protect the processor against these phenomena would have to co-exist on the same chip. Therefore, it is important to understand how silicon reliability is affected by these interactions and develop practical solutions to effectively balance between performance, soft error coverage, and lifetime reliability. Towards this end, this thesis makes the following contributions:

- We analyze the impact of a widely studied RMT design, namely, Simultaneous and Redundantly Threaded (SRT) [19] processors, on lifetime reliability. We find that RMT can cause 3%-17% reduction in the lifetime reliability across the 26 SPEC CPU2000 benchmarks. We find that thermal cycling in the package to be the main reason for this degradation in lifetime reliability.

- We first analyze whether we can leverage Dynamic Voltage Scaling (DVS) that is already available in most processors today to alleviate the lifetime reliability impact. We find that DVS can improve the lifetime reliability of an RMT-based processor by 10%-28% but leads to a severe degradation in performance.

- We then explore the use of a temperature-tracking "partial" RMT [12] scheme where the redundant thread is disabled in response to elevated operating tem-

peratures. Our partial RMT scheme improves performance of redundant execution by 13%-29% but only slightly improves the lifetime reliability of RMT and significantly reduces the soft error coverage.

- Finally we present a hybrid scheme that combines DVS with partial RMT. We show that this hybrid approach provides improvements in the lifetime reliability of RMT comparable to DVS, delivers performance comparable to partial RMT and also provides soft error coverage comparable to fully redundant execution.

The organization of the rest of the thesis is as follows. The next chapter discusses the related work and the experimental methodology is described in Chapter 3. The experimental results are presented in Chapter 4 and Chapter 5 concludes this thesis.

Chapter 2

Background and Related Work

In this Chapter, we review key related work on soft errors and lifetime reliability.

2.1 Soft Errors

Particle-induced soft errors occur due to the interaction of external particles, typically high-energy neutrons, with silicon. Although considered a problem for only large memory arrays (e.g., caches) in the past, soft errors now pose a challenge for even the latches and logic within the processor core [24]. At the architecture level, the soft error vulnerability of the various structures within a processor can be analyzed using the Architecture Vulnerability Factor (AVF) methodology [14, 6]. The AVF quantifies the probability that a fault in a structure within the processor will manifest itself as an error in the externally visible state and takes into account the fact that not all bits that flow through the processor would necessarily affect the Architecturally Correct Execution (ACE).

Protection against soft errors at the architecture level can be provided using coding techniques, and also through spatial and temporal redundancy. A widely studied form of temporal redundancy is Redundant Multi-Threading (RMT) [19], wherein a single program is replicated into two or more redundant threads that execute in-

dependently, for example on the multiple thread-contexts of an SMT processor or on the cores of a multicore processor. The outputs of these redundant threads are compared and any mismatch in their values is used to flag an error. All the hardware structures that lie between the input-replication and output-comparison points, which constitute what is called the "Sphere of Replication" (SoR), are protected through redundant execution whereas those structures that lie outside the SoR are protected through other means, such as ECC or spatial redundancy. Although RMT is an effective technique to detect soft errors, it imposes significant performance overheads. Addressing the performance overheads of RMT has been an active area of research in recent years. The basic optimization approach has been to use some form of "partial RMT" [20, 12], where the redundant thread is sometimes disabled, thereby trading off soft error coverage for improved performance. The proposed partial RMT mechanisms include exploiting dynamic instruction reuse [15, 11] and using high-confidence speculation as a substitute for redundant execution [16, 17]. There has also been prior work on reducing the bandwidth overheads of RMT [27]. None of these prior works have considered the impact of RMT on the lifetime reliability of the processor.

2.2 Lifetime Reliability

Unlike soft errors, lifetime reliability phenomena (which are also referred to as "hard errors") lead to permanent damage. Hard errors can appear during fabrication in the form of defects or later in the field due to wearout and aging, which can be exacerbated by extreme environmental conditions, such as high temperatures, high voltages, electrostatic discharge, etc. Examples of lifetime reliability phenomena include Electromigration (EM), Stress Migration (SM), Thermal Cycling (TC), Time-Dependent Dielectric Breakdown (TDDB), and Negative Bias Temperature Instability (NBTI).

Analytical models for these lifetime reliability phenomena and a methodology for

their use in architecture simulations are discussed in [29] and [31]. In their Reliability-Aware Micro-Processor (RAMP) framework, the authors present models that express the reliability in terms of the Mean-Time To Failure (MTTF) of a microarchitectural structure for a particular lifetime reliability phenomenon, such as EM or TC. These models calculate the MTTF at any given point of time based on static parameters, such as material-dependent constants, and dynamically varying parameters, such as activity factor and temperature. We now present a brief overview of the RAMP methodology. We use RAMP to evaluate lifetime reliability in our experiments. The interested reader is referred to [29] and [31] for more details about the reliability models and the MTTF calculation methodology.

In RAMP, every microarchitectural structure whose reliability is to be calculated (e.g., ALUs, register-files), is assigned an initial reliability budget (expressed in Failures-in-Time or FIT) for each of the five lifetime reliability phenomena (EM, SM, TC, TDDB, and NBTI). RAMP models only *low-frequency thermal cycling* effects on the package. The initial FIT-value of each structure is calculated based on the total lifetime reliability budget of the processor: FIT_{target}. RAMP assumes that FIT_{target} is evenly distributed across the five reliability phenomena and that the failure rate due to a particular phenomenon for a given structure is proportional to its area. In order to model age-related wearout more accurately, where lower failure rates are seen earlier in the lifecycle of a structure, lognormal distributions are used for the different failure mechanisms. After the initial assignment of the FIT budgets, the reductions in the lifetimes of the different structures at runtime are calculated based on dynamically varying parameters, such as temperature, which can be obtained from an architecture simulator that is augmented with power and thermal models. Finally, RAMP uses Monte Carlo simulation to calculate the MTTF of the entire processor from the lognormal lifetime distributions of the various structures and failure mechanisms.

A variety of architecture level techniques have been proposed for addressing lifetime reliability problems. Srinivasan et al. [29] propose the use of Dynamic Reliability Management to meet lifetime reliability targets. In [31], the authors propose the use of spares for key microarchitectural structures to use in the event of an error and also to exploit the spatial redundancy that is inherent in high-performance microprocessors (e.g., multiple functional units) to provide graceful degradation. In [21], the authors propose to leverage the scan-chain logic within superscalar processors to isolate hard faults. The use of dynamic verification techniques to detect hard errors within the processor core are proposed in [5] and [7]. Recent work has proposed architecture-level solutions to mitigate NBTI problems [32].

Chapter 3

Experimental Setup

Our experiments were carried out via execution-driven simulation using the SimpleScalar 3.0 simulator [9], which we modified to simulate RMT. The design space for RMT is large, depending on which structures are chosen to be included within the SoR. Moreover, RMT could be implemented as multiple threads running within a single core, or as redundant threads running on multiple cores of a multicore processor [13]. In this thesis, we study the Simultaneous and Redundantly Threaded (SRT) [19] RMT scheme, which has been proposed to provide soft error detection by extending the microarchitecture of SMT processor cores. Although multicore processors are gaining popularity in the microprocessor market, several commercially available multicore processors still use hardware multi-threading within each core (e.g., IBM POWER6, Intel Nehalem, SUN Niagara), which can be leveraged to implement SRT-based soft error protection.

In SRT, the L1 cache interfaces serve as the input-replication and output-comparison points. One of the redundant threads in the thread-pair runs ahead of the other and hence these threads are referred to as the "leading" and "trailing" threads respectively. In order to improve performance, SRT uses three additional microarchitectural structures: (i) a Branch Outcome Queue (BOQ), (ii) a Load Value Queue (LVQ), and

(iii) a Store Checking Buffer (SCB). More details about the design of these three structures are given in [19] and [13]. In this thesis, we use the term "RMT" to refer to the overall redundancy approach and the term "SRT" to refer to the specific mechanism that we evaluate. Our SRT processor floorplan is an extension of Alpha 21264 floorplan [25] and includes the LVQ, BOQ, and SCB. The floorplan of the SRT-based processor core that we simulate is shown in Figure 3.1. We place these additional structures in the floorplan so that they are co-located with the other microarchitectural blocks that are accessed in the same pipeline stage or in adjacent stages in order to minimize delay. For example, the BOQ is placed adjacent to the branch predictor since the trailing thread accesses the BOQ in the same pipeline stage that the leading thread accesses the branch predictor.

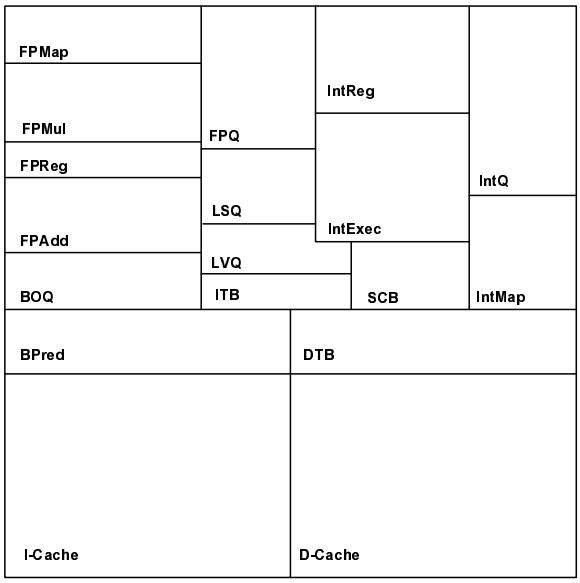


Figure 3.1: SRT Processor Floorplan

We use Wattch [8] for modeling power, HotSpot [25] for simulating the temperature behavior, and the HotSpot leakage power model [26] to calculate leakage power. We sample the temperature of all the structures every 10K cycles during the simulation. We modify SimpleScalar to include the RAMP reliability models [29, 31]. As in prior work [29, 31], we use a FIT_{target} value of 4000, which we distribute across

the various failure modes and we calculate the initial FIT rate due to a particular failure mode for a structure based on its area, as described in Chapter 2.2. The area of various structures within the SoR (which is the region of the chip whose lifetime reliability is most affected by RMT) is given in Table 3.1. We use all 26 benchmarks from the SPEC CPU2000 benchmark suite in our evaluations [28]. The benchmarks were 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 [23]. The parameters of our baseline SRT processor model are given in Table 4.1. Our technology parameters are based on the 65 nm process for which industry data is available.

Structure	**Area (%)**
IntMap	5.79
IntQ	10.73
IntReg	5.75
IntExec	19.13 (2.39 for all ALUs/Multipliers/Dividers)
FPMap	7.03
FPQ	10.53
FPReg	3.99
FPAdd	9.44 (2.36 for all ALUs)
FPMul	4.98 (2.49 for all Multipliers/Dividers)
LVQ	5.88
BOQ	5.88
SCB	5.88

Table 3.1: Area of each structure as a percentage of the area of the structures within the Sphere of Replication

Why Would the Lifetime Reliability of SRT Processors Be Different from SMT Processors? Although both SRT and SMT provide multiple hardware thread contexts, SRT extends the SMT microarchitecture to efficiently support redundant multi-threading by exploiting certain *unique* properties of redundant execution. For example, using an LVQ allows the trailing thread to obtain the value of a load without having to access the cache, whereas it is not generally possible to avoid a cache access on a load for a thread running on an SMT processor. Similarly, the BOQ provides perfect branch-prediction for the trailing thread whereas it is very challenging to provide

perfect branch-prediction in the general case. As [19] quantitatively demonstrates, using these additional structures to build an SRT processor significantly boosts the performance of RMT compared to running redundant threads on an SMT core. This significant boost in performance is the result of a correspondingly higher utilization of the microarchitectural resources within the core [15, 11, 16, 17]. The prior works have looked at the performance impact. In this thesis we examine the lifetime reliability impact.

Chapter 4

Results

Given the rising soft error rates for the logic and latches within the processor core, processors will have to implement mechanisms such as RMT to meet the soft error budgets. In addition to providing adequate soft error coverage it is desirable that RMT-based processors do not degrade single-thread performance (i.e., the performance should be as if there was no redundant thread competing for the hardware resources). It is also desirable that the lifetime reliability of an RMT-based processor not exceed the budget set for earlier (non-redundant) generations of the same processor family. In this thesis, we look at the lifetime reliability issue and use the lifetime reliability of the single-threaded execution mode as a proxy for the desired lifetime reliability. In the first set of results, we quantify this reduction in lifetime reliability due to RMT. We then evaluate two different approaches to buy back this lifetime reliability loss: (i) DVS (Chapter 4.2) and (2) partial RMT (Chapter 4.3). We finally present the results for a hybrid scheme that combines DVS and partial RMT.

We assume a series failure-model for lifetime reliability. This model captures the scenario where the first structure to fail will lead to a complete failure of the processor [31]. We believe that the series failure-model is a reasonable assumption for two reasons. First, one of the primary motivations for RMT is to provide soft error

Technology Parameters	
Process Technology	65 nm
Supply Voltage	1.3 V
Clock Frequency	3 GHz
Ambient Temperature	45 C
Processor Parameters	
Pipeline Width	8
Fetch-Queue Size	16
Branch-Predictor Type	Bimodal, with 2K-entry table
RAS Size	64
BTB Size	2K-entry 4-way
Branch-Misprediction Latency	7
ROB Size	128
LSQ Size	64
Integer ALUs	6
Integer Multipliers/Dividers	2
FP ALUs	4
FP Mult./Div./Sqrt.	2
L1 Cache Ports	4
L1 D-Cache	32 KB 4-way with 32B line-size
L1 I-Cache	32 KB 2-way with 32B line-size
L2 Unified Cache	256 KB 4-way with 64B line-size
I-TLB	128 entries 4-way
D-TLB	256 entries 4-way
TLB Miss-Latency	30 cycles
Main Memory Latency	200 cycles

Table 4.1: Baseline processor configuration

protection for commodity microprocessors where structural duplication techniques, such as the use of spares, may be prohibitive to incorporate due to cost. Second, even in high-end processors, structural duplication is used only to protect components that lie outside the processor core, such as the caches and I/O controllers [18]. Since the SoR of SRT is completely internal to the core, the failure of any structure within the SoR will lead to a failure of the processor as a whole.

4.1 Impact of Redundant Multi-Threading on Processor Lifetime Reliability

We first test the hypothesis that the increased level of activity within the processor due to redundant execution could accelerate the wearout of the structures that they stress. We conduct an experiment where we analyze the reduction in the processor lifetime reliability due to redundant execution for the SPEC CPU2000 benchmarks.

The results from this experiment are given in Figure 4.1. Each bar in Figure 4.1 corresponds to a particular benchmark and shows the reduction in lifetime reliability due to SRT with respect to the single-threaded execution mode. We can see that SRT can reduce the lifetime reliability by 3%-17%, with an average lifetime reduction of 7% across all 26 benchmarks. The impact on reliability is especially strong for the integer benchmarks, whose average lifetime reduction is 11%, whereas the degradation for floating-point workloads is 4%. In general, across all the benchmarks, we find that the main cause for lifetime reliability is thermal cycling in the package due to high operating temperatures in the integer register-file. We now explain the reasons for these trends.

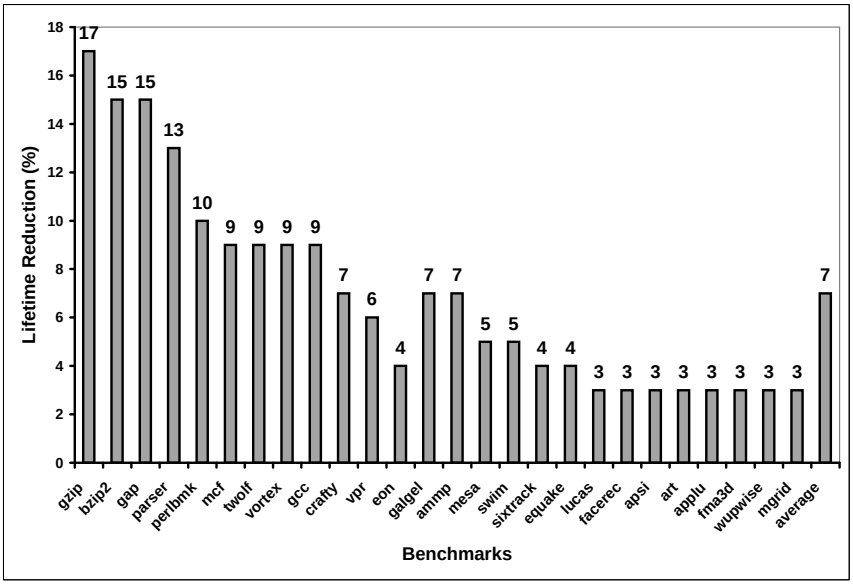


Figure 4.1: Degradation in processor lifetime reliability due to SRT.

The MTTF due to thermal cycling, $MTTF_{TC}$, is given by the equation:

$$MTTF_{TC} \propto (\frac{1}{T - T_{ambient}})^q$$

where, T and $T_{ambient}$ are the average temperature of the structure and the ambient temperature respectively, and q is the Coffin-Manson exponent [29]. As in the prior work on RAMP [29], we use a q value of 2.35 for the package. As the equation indicates, high operating temperatures have a significant impact on TC due to the exponential relationship between temperature and $MTTF_{TC}$. In our experiments we find the integer register-file to be the hottest structure for most of the benchmarks, a result that concurs with previous studies on the thermal behavior of processors [25] and leads to a significant amount of thermal cycling. Interestingly, we find that thermal cycling due to hotspots in the integer register-file to be the primary cause for processor lifetime degradation even for the floating-point benchmarks. In order to determine why this happens, we profiled the benchmarks to determine their instruction mix. The breakdown of the instruction mix for the benchmarks, classified into integer and floating-point instructions, are given in Table 4.2. The classification of load and store instructions into the two categories are based on whether they operate on integer or floating-point registers. As we expect, a large fraction (82.4% on average) of the instructions in the integer benchmarks are indeed integer instructions. As a result, there are a large number of accesses to the integer register-file, which increases the temperature of this structure. The `eon` benchmark experiences the least degradation in lifetime reliability. This is due to the presence of a significant number of floating-point instructions, as shown in Table 4.2(a), which reduces the number of physical accesses to the integer register-file.

As we can see in Table 4.2(b), several floating-point benchmarks also have a considerable number of integer instructions and therefore these benchmarks also exercise the integer register-file. In fact, `mesa`, `art`, and `apsi` have a significantly higher number of integer instructions than floating-point instructions. We find some interesting behaviors for few floating-point benchmarks like `mgrid`, `lucas` and `sixtrack`. We find that some structures for these benchmarks experience the highest temperature. The

hottest units for these benchmarks are floating-point structures (floating-point adder or register-file) while the second hottest unit is the integer register-file. From Table 4.2 we see that these benchmarks have significantly higher number of floating-point instructions. But the main cause of lifetime degradation for these benchmarks is thermal cycling due to the integer register-file. Recall from Chapter 2.2, the MTTF of the processor depends on both area and dynamically varying parameters like the temperature of the structures. Therefore, since the area of the floating-point units are much smaller than the integer register-file, the integer register-file has more impact on the lifetime degradation. However, since the overall percentage of integer instructions in the integer benchmarks is significantly higher than in the floating-point benchmarks, the integer benchmarks impact lifetime reliability to a greater extent, as shown in Figure 4.1.

To summarize, thermal cycling due to high operating temperatures in the integer register-file is the primary reason for degradation in the lifetime reliability due to SRT. Although SRT is effective in providing protection against soft errors, it has a detrimental effect on lifetime reliability. We now explore two different approaches to mitigate this problem.

4.2 Improving Lifetime Reliability through Dynamic Voltage Scaling

Dynamic Voltage Scaling (DVS) is a standard power/temperature management feature found in processors today. DVS can provide a cubic reduction in the power density, which can help reduce the temperature of the chip. However, since DVS scales both the voltage and the clock frequency, this reduction in power comes at the expense of performance. We now study whether we can simply leverage DVS to reduce thermal cycling in SRT-based processors.

Benchmark	Instruction Mix (%)	
	Integer Instructions	Floating-Point Instructions
gzip	88.27	0.0
bzip2	85.74	0.09
gap	87.67	0.0
parser	83.96	0.0
perlbmk	86.25	0.1
mcf	79.0	1.06
twolf	82.98	4.81
vortex	82.3	0.1
gcc	86.65	0.0
crafty	88.5	0.1
vpr	72.98	16.57
eon	64.3	24.01
Average	**82.4**	**3.9**

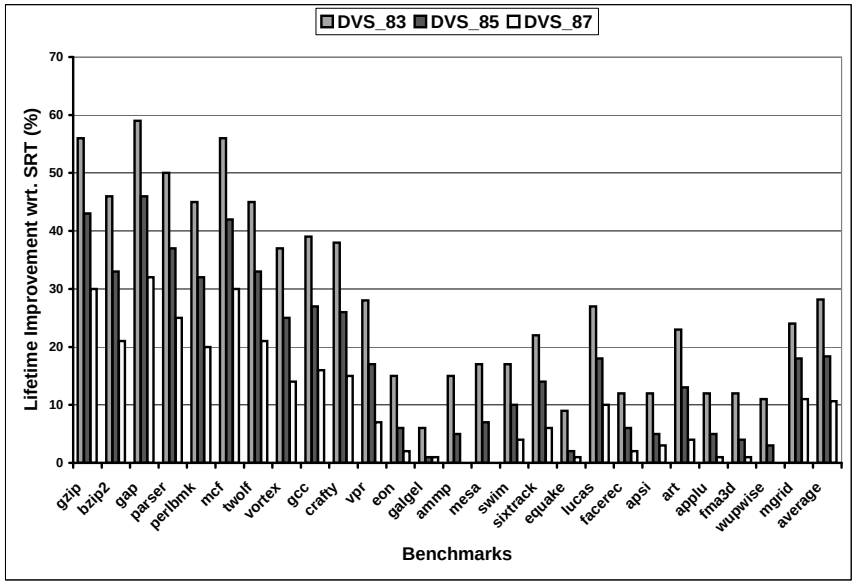
(a) Integer Benchmarks

Benchmark	Instruction Mix (%)	
	Integer Instructions	Floating-Point Instructions
galgel	48.51	46.18
ammp	44.44	47.59
mesa	72.29	18.95
swim	34.86	64.66
sixtrack	18.26	79.85
equake	36.54	59.48
lucas	26.5	72.92
facerec	48.64	45.92
apsi	50.24	46.71
art	52.32	33.94
applu	17.1	82.22
fma3d	45.08	50.85
wupwise	49.06	40.98
mgrid	12.71	87.0
Average	**39.8**	**55.5**

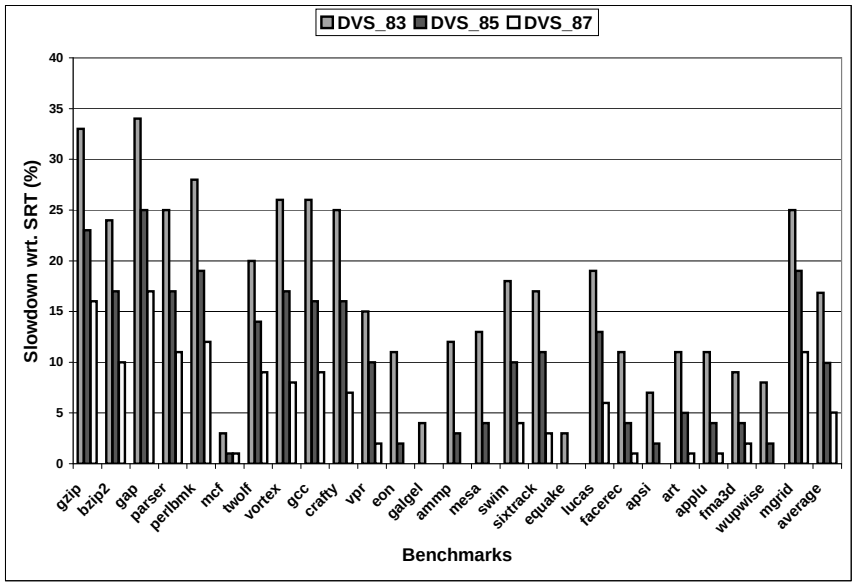
(b) Floating-Point Benchmarks

Table 4.2: Percentage of integer and floating-point instructions in the benchmarks.

Our DVS scheme is based on the feedback-based approach proposed by Skadron et al. [25], which uses a Proportional-Integral (PI) controller to manage temperature. This scheme works as follows. Given a temperature threshold, which is the maximum temperature within which we would like to operate all the on-chip structures, and a gain value, the difference between the current temperature and the threshold value is used to set the scaling factor. As in [25], we use a gain value of 10, allow the DVS settings to vary in steps from 50% to 100% of the nominal voltage, and use a low-pass filter to the controller output to prevent frequent DVS actions due to temperature fluctuations. Note that although reducing the voltage could increase the soft error rate (SER) per bit, the range of voltages used by our DVS scheme has negligible impact on the SER for the 65nm process technology [22].



(a) Impact on lifetime reliability



(b) Impact on performance

Figure 4.2: Impact of Dynamic Voltage Scaling on lifetime reliability and performance.

A key parameter that determines the temperature at which an unit operates is the temperature threshold at which DVS is triggered. Higher the threshold value, lower is the likelihood of triggering DVS, which can in turn provide higher performance. However, this higher performance comes at the cost of decreased lifetime reliability. On the other hand, lowering the threshold value to improve lifetime reliability could lead to significant performance loss. In order to analyze these tradeoffs, we evaluate DVS with three threshold values: 83 C, 85 C, and 87 C respectively. The results from these experiments are given in Figure 4.2. Figure 4.2(a) quantifies the improvement in lifetime reliability with DVS and Figure 4.2(b) gives the corresponding performance impact. The bars for DVS_{83}, DVS_{85}, and DVS_{87} correspond to the three temperature thresholds. As Figure 4.2(a) indicates, the use of DVS provides significant improvements in lifetime reliability for the integer benchmarks. The DVS_{83}, DVS_{85}, and DVS_{87} schemes provide 42.3%, 30.1%, and 18.8% improvements in lifetime reliability respectively. However, these improvements come at the cost of a significant slowdown in performance as shown in Figure 4.2(b). The `mcf` benchmark shows an interesting behavior. `mcf` gains large improvements in lifetime reliability but suffers very little performance degradation over the non-DVS based SRT system. We find that `mcf` has the lowest IPC (0.397) among all the benchmarks. This is due to the fact that `mcf` experiences a significantly higher miss-rate in the L1 data cache than the other benchmarks. The L1 D-cache miss-rate for `mcf` is 41.2%, which is significantly higher than that for the benchmark with the second highest miss-rate - `gzip` - whose L1 D-cache miss-rate is 12%. As a result, `mcf` experiences a larger number of stalls and therefore grossly underutilizes the processor resources, thereby resulting in a cooler temperature profile. This, in turn, causes DVS to be triggered less often. However, during those occasional short-duration periods when the `mcf` workload does elevate the operating temperature, we find that the temperatures can be high enough to impact the lifetime reliability. Engaging DVS during these short intervals reduces

the operating temperature thereby improving lifetime reliability. Although `gap` shows a lifetime improvement trend similar to `mcf`, the IPC of the former is much higher (1.66) and therefore incurs a higher performance penalty than `mcf` due to DVS as shown in Figure 4.2(b).

As we expect from the results in Chapter 4.1, the lifetime improvements and the corresponding performance losses for the floating-point benchmarks are lower than those for the integer benchmarks. The three highest lifetime improvements as well as the greatest performance losses are observed for `mgrid`, `lucas` and `sixtrack`. These are the floating-point benchmarks with the hottest structures as mentioned in Chapter 4.1. As DVS directly controls the temperature, these benchmarks experience the highest lifetime improvement. However, since DVS gets trigerred more often for these benchmarks, they experience a much higher performance loss.

Between the three DVS schemes, we can observe that the use of progressively higher temperature thresholds results in larger drops in lifetime reliability whereas the performance trends scale more or less linearly with the thresholds. The reason for this is as follows. For all three DVS schemes, we find that TC is still the primary reason for degradation in lifetime reliability. Since $MTTF_{TC}$ is proportional to $(\frac{1}{T-T_{ambient}})^{2.35}$, a linear change in temperature has an exponential impact on TC. On the other hand, temperature and frequency are related *linearly* [10]. Since performance depends on the frequency, the performance curves show a linear variation with temperature for the DVS schemes. With the use of higher temperature thresholds, the effectiveness of DVS starts diminishing. Although the integer benchmarks still benefit from the DVS_{87} scheme, most floating-point benchmarks get negligible improvements in lifetime reliability (since their operating temperatures remain below the trigger threshold temperature) and consequently experience little slowdown in performance.

Summary: There are merits and demerits to using DVS. On the positive side, on av-

erage, it improves lifetime reliability to a larger extent than it degrades performance, although the gap between the two narrows at higher temperatures. The average lifetime improvements across the entire benchmark suite for the DVS_{83}, DVS_{85}, and DVS_{87} schemes are 28%, 18%, and 10% respectively, whereas the corresponding performance degradations are 16%, 10%, and 5% respectively. On the other hand, since SRT itself imposes significant performance penalties, the additional slowdown due to DVS could lead to unacceptably low single-thread performance. Choosing higher temperature thresholds to offset these performance losses can diminish the effectiveness of DVS in improving lifetime reliability. Therefore, although DVS is a candidate solution to tackle lifetime reliability in SRT processors and is a standard feature in most processors today, we cannot directly use it due to its negative performance impact.

4.3 Improving Lifetime Reliability through Partial Redundant Multi-Threading

Partial RMT is a technique to address the performance impact of redundant execution. In partial RMT, parts of the redundant thread are not executed to trade off soft error coverage for improved performance [15, 11, 16, 17, 33]. From the lifetime reliability perspective, the removal of instructions from the SoR can reduce the contention for structures such as the integer register-file, which could in turn lower temperatures and hence reduce TC. Therefore, it is interesting to explore whether partial RMT could be a viable approach to balance soft error coverage with lifetime reliability.

We evaluate the performance, soft error coverage, and the lifetime reliability impact of a *temperature-tracking partial RMT* scheme. This scheme works as follows. Similar to DVS, our partial RMT scheme also uses a temperature threshold. At the start of the execution, we run the processor in SRT mode through the thermal and

performance warmup phases (as described in Chapter 3), then continue redundant execution for another 10K cycles, and finally measure the temperatures of all the structures. (Recall that 10K cycles is our temperature sampling interval). If the maximum temperature of all the components is below the threshold value, we continue the fully redundant execution and therefore get complete soft error coverage. However, when the maximum temperature exceeds the threshold, we disable SRT and switch to the single-threaded execution mode until the temperature falls below the threshold. Once the temperature goes below the threshold, we re-enable SRT. Therefore, during periods when SRT is disabled, soft error coverage is compromised. As with DVS, we use a low-pass filter to avoid frequent switches between the SRT and single-threaded mode due to temperature fluctuations. In the case of DVS, the selection of a particular temperature threshold affects only lifetime reliability and performance. In partial RMT, the threshold value also affects soft error coverage, in particular the Architectural Vulnerability Factor (AVF) of the structures within the SoR [14, 6]. In our experiments, we use the same three temperature thresholds as before: 83 C, 85 C, and 87 C, and we refer to the corresponding partial RMT policies as $PRMT_{83}$, $PRMT_{85}$, and $PRMT_{87}$ respectively. The improvement in lifetime reliability with respect to SRT for these partial RMT policies is given in Figure 4.3(a). Unlike DVS, turning off redundant execution always provides a performance benefit and therefore we plot the performance improvement, rather than the slowdown, of partial RMT with respect to SRT in Figure 4.3(b). In Table 4.3, we present the average AVFs of the Reorder Buffer (ROB), Load/Store Queue (LSQ), Issue Queue (ISQ), which includes both the integer and floating-point queues, and the register files (Regs). We present the AVFs of these structures when the processor runs in the single-threaded mode, as well as those of the three partial RMT policies. A higher AVF value indicates that the given structure is more vulnerable to soft errors and the goal of SRT is to reduce the AVFs to zero via redundant execution. From Figure

4.3, we can see that the use of partial RMT results in significant performance gains (28.6%, 23.4%, and 13% for $PRMT_{83}$, $PRMT_{85}$, and $PRMT_{87}$ respectively). However, partial RMT provides only a small benefit in terms of lifetime reliability. The three policies provide an average lifetime reliability improvement of only 7.3%, 6.7%, and 5.5% with respect to SRT, which is significantly less than the benefits provided by DVS for the same temperature threshold values. For the integer workloads, the lifetime reliability and performance trends remain invariant across all three policies, except for `eon`.

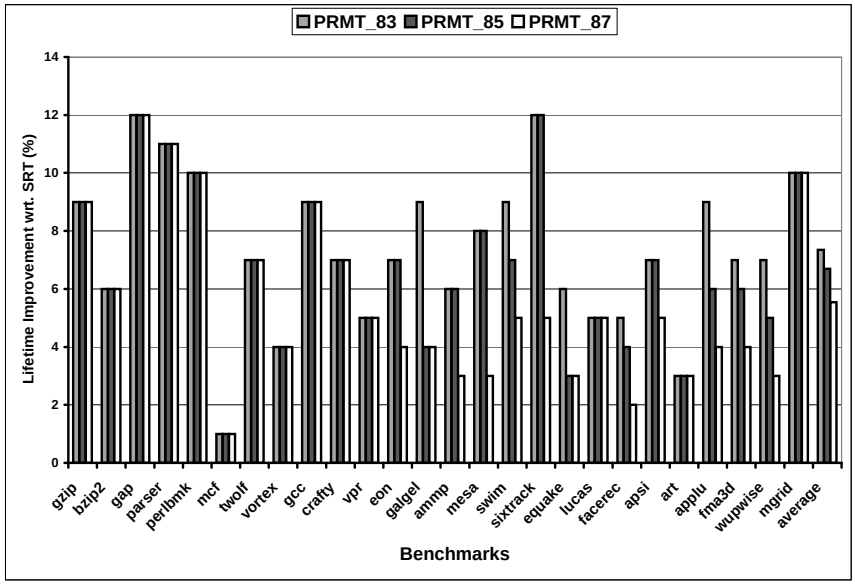
Benchmark	$Single-Threaded$				$PRMT_{83}$				$PRMT_{85}$				$PRMT_{87}$			
	ROB	LSQ	ISQ	Regs	ROB	LSQ	ISQ	Regs	ROB	LSQ	ISQ	Regs	ROB	LSQ	ISQ	Regs
gzip	48	25	19	32	46	25	17	29	46	25	17	29	46	25	17	29
bzip2	34	24	17	21	34	25	16	20	34	25	16	20	34	25	16	20
gap	27	21	14	17	26	21	13	15	26	21	13	15	26	21	13	15
parser	42	23	15	29	42	24	14	27	42	24	14	27	42	24	14	27
perlbmk	18	17	11	11	17	16	10	10	17	16	10	10	17	16	10	10
mcf	51	44	15	25	51	36	14	28	51	36	14	28	51	36	14	28
twolf	51	34	23	33	50	34	22	32	50	34	22	32	50	34	22	32
vortex	67	56	46	42	68	55	46	42	68	55	46	42	68	55	46	42
gcc	17	24	7	8	19	18	10	10	19	18	10	10	19	18	10	10
crafty	25	19	14	15	25	19	14	14	25	19	14	14	25	19	14	14
vpr	56	49	17	38	56	49	17	37	56	49	17	37	56	49	17	37
eon	23	22	17	17	23	22	16	16	23	22	16	16	0	0	0	0
galgel	72	48	62	60	27	23	20	19	0	0	0	0	0	0	0	0
ammp	83	60	54	63	83	66	51	59	83	66	51	59	0	0	0	0
mesa	68	49	50	47	69	53	48	45	69	53	48	45	0	0	0	0
swim	88	50	84	74	88	49	81	71	64	36	59	52	30	15	27	24
sixtrack	87	43	64	71	87	43	62	70	87	43	62	70	29	14	22	24
equake	93	87	52	69	93	86	52	68	5	5	3	4	0	0	0	0
lucas	100	41	48	83	100	41	48	73	100	41	48	73	79	35	40	65
facerec	84	58	74	70	84	58	73	68	78	55	63	63	24	17	20	19
apsi	56	43	41	46	56	43	40	45	56	43	40	45	0	0	0	0
art	67	40	53	53	67	48	49	49	67	48	49	49	67	48	49	49
applu	86	66	67	70	84	66	66	70	27	23	21	23	6	5	5	5
fma3d	60	56	49	47	61	57	48	46	29	26	23	22	5	4	4	4
wupwise	56	31	49	46	56	34	46	43	26	16	22	20	0	0	0	0
mgrid	98	72	91	82	98	72	90	80	98	72	90	80	98	72	90	80

Table 4.3: Architectural Vulnerability Factors of key structures within the Sphere of Replication in the single-threaded mode and for the $PRMT_{83}$, $PRMT_{85}$, and $PRMT_{87}$ partial RMT policies.

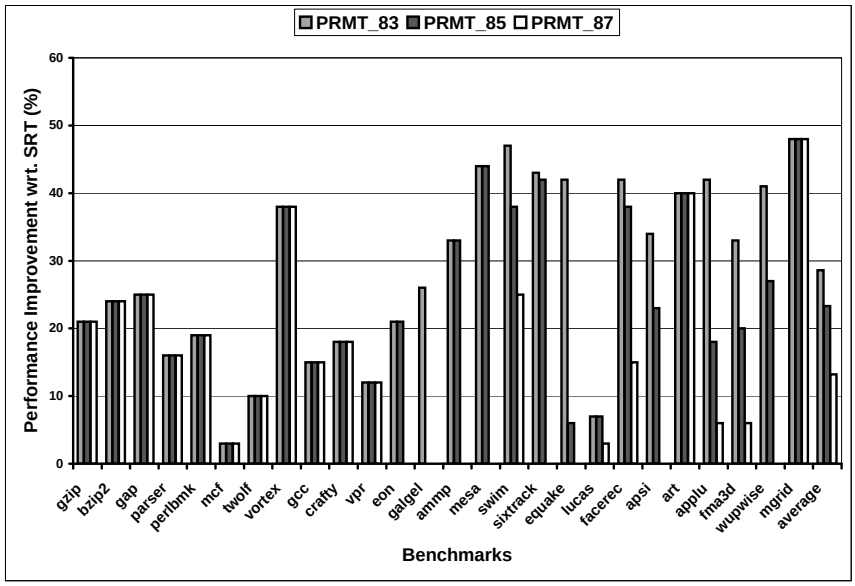
For all three temperature thresholds, we find that redundant execution is disabled most of the time for the integer benchmarks in response to their high operating temperatures. Therefore, the use of partial RMT significantly compromises the soft error coverage for these workloads and the AVFs do not decrease with the use of higher threshold values, as shown in Table 4.3. However, since the processor is protected via SRT for the first 10K cycles, the AVFs for the partial RMT policies are slightly lower than those for the single-threaded mode. A few of the workloads (e.g., the AVF of the LSQ for `bzip2`) in Table 4.3 show a small increase in the AVF of certain structures

for the partial RMT modes. This counter-intuitive result is an artifact of how we compute the average AVF. In reality, at the end of a SimPoint, there are typically a few instructions whose ACE-ness is *unknown* because the benchmark is not allowed to run to completion and therefore we cannot determine what impact, if any, those instructions would have on the architected state of the machine [6]. Therefore, based on whether we assume these instructions to be ACE or un-ACE, the AVF would vary over a range. Since we found these AVF ranges to be small, we merely present the average over each such range. The small difference in the AVF values between the single-threaded and partial-RMT modes in Table 4.3 is within these small ranges and therefore correspond to roughly equivalent AVF values.

Several floating-point benchmarks switch back and forth between the redundant and single-threaded execution modes and show prominent reductions in their AVF values as we go in for higher temperature thresholds. In fact, the AVFs of several floating-point benchmarks drop to zero for $PRMT_{87}$ as a result of their operating temperature being below the threshold value most of the time and therefore not requiring SRT to be disabled. However, the AVFs of the integer benchmark `eon` also show sensitivity to partial RMT. We now provide a more detailed explanation for these results. We can observe that the lifetime reliability benefits of using partial RMT (Figure 4.3(a)) are comparable to the degradation in lifetime reliability of the processor due to SRT (Figure 4.1) for several integer benchmarks. As mentioned previously, the integer benchmarks operate in single-threaded mode most of the time due to their high operating temperatures. However, we can see that the bars in Figure 4.3(a) are consistently lower than those in Figure 4.1, except for `eon`. This is because we run the processor in SRT mode through the thermal and performance warmup phases and for the first 10K cycles of execution, during which time the temperature rises. The temperature rise is especially sharp for benchmarks such as `gzip`, `bzip2`, `vortex`, and `mcf`, which we find undergo a higher amount of thermal cycling during



(a) Impact on lifetime reliability



(b) Impact on performance

Figure 4.3: Impact of partial RMT on lifetime reliability and performance.

those first 10K cycles of redundant execution. Although SRT is subsequently disabled for these benchmarks, the overall lifetime reliability of the processor is impacted during this initial part of the execution.

The `eon` benchmark and the floating-point workloads show a different trend. We find that these workloads show improvements in their lifetime reliability to a greater extent than they show reductions in their lifetime reliability due to SRT. As we can see in Figure 4.3(a), the bars for these workloads are higher than the corresponding bars in Figure 4.1. In order to understand why this happens, we analyzed the pattern of accesses to the integer register-file for all the workloads. As Table 4.2 shows, the floating-point benchmarks have varying degrees of instructions in their instruction mix and `eon` has a sizable number of floating-point instructions as well. At runtime, we find that the accesses to the integer register-file tend to occur in clusters of consecutive integer instructions in the single-threaded mode. In SRT mode, these clusters are larger due to integer instructions from both the leading and trailing threads accessing the integer register-file, thereby affecting its temperature. In the case of partial RMT, where the processor moves back and forth between the single-threaded and SRT modes, we find that the integer instructions get interspersed with floating-point instructions. As a result of this, the number of consecutive accesses to the integer register-file are lower in partial RMT mode than in both the single-threaded and SRT modes. This pattern of accesses has a "cooling" effect on the integer register-file and hence these benchmarks show a greater improvement in their lifetime reliability. In general for partial RMT, we find that the performance improvements for the floating-point benchmarks are higher than those for the integer benchmarks. The reason behind this is that, during the SRT mode, the floating-point benchmarks experience higher performance loss. In the SRT mode, both the leading and trailing threads have to be completed in order to commit the instructions. The floating-point benchmarks have a significant number of high-latency floating-point

instructions, and therefore the instructions those are data-dependent on those high-latency instructions will wait for a longer time in the issue queue. As a result, in the SRT mode both the leading and trailing threads have to wait for a longer time, leading to performance loss. Consequently, the floating-point benchmarks enjoy more performance improvements as a result of single-threaded execution in partial RMT. One exception among the floating-point benchmarks is `lucas` which experiences the least performance improvement. We find that this benchmark has comparable IPC in both the single-threaded and SRT modes due to the the effective interleaving of the instructions from the leading and trailing thread.

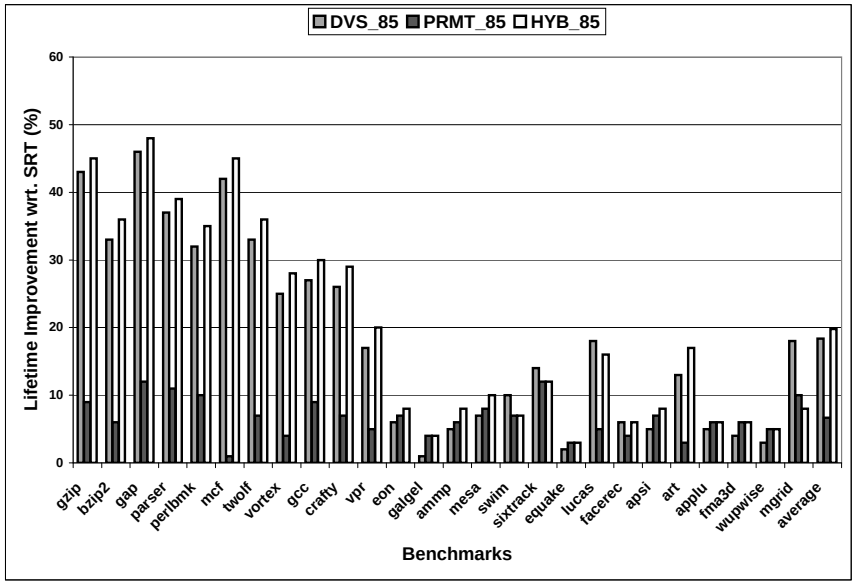
Summary: Between DVS and partial RMT, we find that the lifetime reliability benefits obtained by modulating voltage and frequency trumps toggling of the redundant execution. This trend is especially pronounced for the integer benchmarks, where the operating temperature in even the single-threaded mode is quite high. Although disabling SRT lowers temperatures by a small amount, DVS is a much more effective knob to manage temperature and lifetime reliability. Moreover DVS can improve lifetime reliability without compromising soft error coverage, whereas partial RMT reduces soft error coverage. However, DVS degrades single-thread performance whereas partial RMT can boost single-thread performance. Given these relative strengths and weaknesses of DVS and partial RMT, it would be interesting to combine both techniques such that we can better balance lifetime reliability with soft error protection and also get good single-thread performance. We now explore one such hybrid technique.

4.4 Combining DVS with Partial RMT

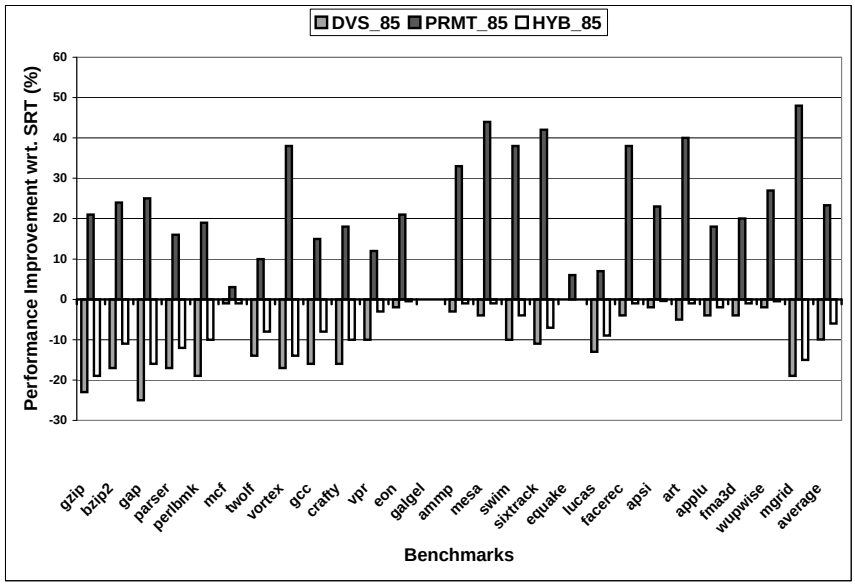
The goal of our hybrid scheme is to effectively respond to temperature violations while attempting to maximize performance. In order to accomplish this, we leverage

the unique benefits that DVS and partial RMT each provide. That is, we use DVS as the primary knob for controlling temperature and use partial RMT to boost performance. We implement this hybrid technique as follows. As with the two previous approaches, a control action is initiated in response to the operating temperature crossing a given temperature threshold value. When a temperature violation is detected, we first disable SRT. We then allow the processor to run in the single-threaded mode and measure the temperature every 10K cycles. If the operating temperature drops below the threshold, we re-enable SRT without having to invoke DVS. However if the operating temperature does not drop below the threshold for three successive measurement intervals (i.e., 30K cycles), we invoke the DVS controller and select the appropriate scaling factor based on the most recent temperature measurement. Once voltage and frequency have been scaled, we allow the processor to run with SRT disabled. Therefore, during the interval between when the response is initiated and when the next temperature measurement indicates that the temperature has dropped below the threshold, the DVS mechanism stays engaged. The performance loss due to operating the processor at the lower frequency is offset to some degree by running in the single-threaded mode, but at the expense of reduced soft error coverage. Once we detect that the temperature has fallen below the threshold, we scale up the voltage and frequency appropriately and re-enable SRT.

We have evaluated our hybrid scheme for all three temperature thresholds, but present the graphs only the 85 C threshold due to space reasons. To summarize the results across the three thresholds, we find that the hybrid scheme provides 12.6%-29.3% improvement in lifetime reliability over SRT while slowing down performance by 3.3%-12.8% compared to SRT. The lifetime improvement and performance impact of the hybrid scheme, which we refer to as HYB_{85}, are given in Figures 4.4(a) and 4.4(b). Each set of bars in the graphs presents the results for HYB_{85} alongside those of DVS_{85} and $PRMT_{85}$ to show the comparative trends. The AVFs of the



(a) Impact on lifetime reliability



(b) Impact on performance

Figure 4.4: Impact of the hybrid partial RMT scheme for the 85 C temperature threshold.

structures within the SoR for all three schemes along with those of the single-threaded execution are given in Table 4.4. From Figure 4.4(a), we can see that HYB_{85} provides good improvement in lifetime reliability across the entire benchmark suite (19.8% on average). This improvement is better than that provided by DVS or by partial RMT individually. Although this lifetime improvement comes with a performance penalty (5.9% on average), this penalty is less severe than DVS, as shown in Figure 4.4(b). We can see in Table 4.4 that the HYB_{85} scheme significantly reduces the AVFs of all the structures, providing either complete, or nearly complete, coverage against soft errors for almost all the benchmarks. In the case of the other two temperature thresholds, we found that the AVFs of all the structures show a similar trend and the hybrid approach provides good coverage against soft errors.

Benchmark	$Single-Threaded$				$PRMT_{85}$				HYB_{85}			
	ROB	**LSQ**	**ISQ**	**Regs**	**ROB**	**LSQ**	**ISQ**	**Regs**	**ROB**	**LSQ**	**ISQ**	**Regs**
gzip	48	25	19	29	46	25	17	29	10	6	5	6
bzip2	34	24	17	21	34	25	16	20	7	5	4	4
gap	27	21	14	17	26	21	13	15	7	5	3	4
parser	42	23	15	29	42	24	14	27	8	4	3	5
perlbmk	18	17	11	11	17	16	10	10	7	7	4	4
mcf	51	44	15	25	51	36	14	28	6	5	3	2
twolf	51	34	23	33	50	34	22	32	7	6	3	2
vortex	67	56	46	42	68	55	46	42	15	12	11	9
gcc	17	24	7	8	19	18	10	10	2	2	1	1
crafty	25	19	14	15	25	19	14	14	5	4	3	3
vpr	56	49	17	38	56	49	17	37	7	6	2	4
eon	23	22	17	17	23	22	16	16	1	1	1	1
galgel	72	48	62	60	0	0	0	0	0	0	0	0
ammp	83	60	54	63	83	66	51	59	3	2	2	2
mesa	68	49	50	47	69	53	48	45	4	3	3	3
swim	88	50	84	74	64	36	59	52	12	6	11	10
sixtrack	87	43	64	71	87	43	62	70	11	5	10	10
equake	93	87	52	69	5	5	3	4	0.08	0.06	0.06	0.06
lucas	100	41	48	83	100	41	48	73	14	9	9	11
facerec	84	58	74	70	78	55	63	63	4	3	3	3
apsi	56	43	41	46	56	43	40	45	4	3	2	3
art	67	40	53	53	67	48	49	49	7	5	5	5
applu	86	66	67	70	27	23	21	23	10	8	8	8
fma3d	60	56	49	47	29	26	23	22	2	2	1	1
wupwise	56	31	49	46	26	16	22	20	1	1	1	1
mgrid	98	72	91	82	98	72	90	80	17	13	16	14

Table 4.4: Architectural Vulnerability Factors of key structures within the Sphere of Replication for the HYB_{85} policy. The single-threaded mode and $PRMT_{85}$ AVFs are shown to facilitate data comparison.

In terms of lifetime reliability, the integer benchmarks derive the most benefit from the hybrid scheme. This is due to the fact that these benchmarks trigger both partial

RMT and DVS to reduce temperature and the combination of these two techniques yields the best improvements in lifetime reliability. Although engaging DVS results in a significant slowdown, some of this performance loss is offset by the fact that SRT is disabled and the processor runs in the single-threaded mode. Moreover, since both partial RMT and DVS are engaged in response to a temperature emergency for these benchmarks, the window of time for which the processor runs in single-threaded mode is much shorter than in the $PRMT_{85}$ approach. As a result the AVFs of the integer benchmarks are significantly lower with HYB_{85} than with $PRMT_{85}$, as shown in Table 4.4. The performance slowdown for `mcf` is much lower than those for the other integer benchmarks for the same reasons as discussed in Chapter 4.2.

In general, most floating-point benchmarks break even (or nearly break even) with the performance of SRT since they operate at a lower temperature than the integer benchmarks and trigger the thermal management mechanism less often. However, there are variations in the lifetime improvement characteristics as well as the performance behavior across the floating-point benchmarks. In case of `ammp`, `mesa`, `apsi`, and `art`, HYB_{85} provides the best improvement in lifetime reliability. The operating temperatures for this workload run below the threshold value of 85 C most of time. However, when a temperature emergency does occur, we find that both the partial RMT and the DVS mechanisms are engaged, thereby providing good improvements in lifetime reliability. For `swim`, `sixtrack`, `lucas`, and `mgrid`, HYB_{85} provides less improvement than DVS_{85} and incurs a significant performance slowdown. These benchmarks cause the processor to operate at a higher temperature, as a result of which DVS remains engaged for a longer duration of time and disabling SRT does not adequately offset the slowdown in performance over this long period. Moreover, since the processor operates in the single-threaded mode during this long time interval, these workloads also get less soft error coverage, as indicated by the higher AVF values for these benchmarks in Table 4.4. The lifetime reliability improvement for

`equake`, `applu`, `fma3d`, and `wupwise` with HYB_{85} are comparable to $PRMT_{85}$. For these benchmarks, we find that disabling SRT by itself provides most of the required reduction in temperature and DVS is engaged only briefly to bring the temperature below the threshold value.

Chapter 5

Conclusions and Future Work

Silicon reliability is one the key challenges facing the microprocessor industry. Architects have to design processors that are resilient against soft errors and lifetime reliability, while still delivering high performance to applications and users. Although a large body of research exists on tackling soft errors and lifetime reliability individually, there has been little work on how reliability mechanisms developed to address one type of reliability problem might impact other aspects of silicon reliability. In this thesis, we explore how Redundant Multi-Threading (RMT), a mechanism for protecting processors against soft errors, affects lifetime reliability. We evaluate three different approaches to mitigate this problem, namely, Dynamic Voltage Scaling (DVS) that is available in processors today, partial RMT, and a hybrid scheme which utilizes both DVS and partial RMT. Each approach has certain strengths and weaknesses with respect to performance, soft error coverage, and lifetime reliability. In future work, we plan to explore how one could use information about the actual wearout of various microarchitectural structures using hard error sensors [3, 4] and AVF predictors [33, 1] to be used to balance between these different figures of merit. We also plan to study how other tunable partial redundancy techniques [16, 2] can be used in conjunction with these sensors to craft reliability management policies.

Bibliography

[1] A. Biswas,N. Soundararajan,S.S. Mukherjee and S. Gurumurthi. Quantized AVF: A Means of Capturing Vulnerability Variations over Small Windows of Time. In *IEEE Workshop on Silicon Errors in Logic - System Effects*, March 2009.

[2] B.C. Sutton and S. Gurumurthi. Single-Threaded Mode AVF Prediction During Redundant Execution. In *IEEE Workshop on Silicon Errors in Logic - System Effects*, March 2009.

[3] A.C. Cabe, Z. Qi, S.N. Wooters, T.N. Blalock and M.R. Stan. Small embeddable NBTI sensors (SENS) for tracking on-chip performance decay. In *International Symposium on Quality Electronic Design*, page 1–6, 2009.

[4] E. Karl, P. Singh, D. Blaauw and D. Sylvester. Compact in situ sensors for monitoring nbti and oxide degradation. In *IEEE International Solid-State Circuits Conference*, pages 410–623, February 2008.

[5] T. Austin. DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 196–207, November 1999.

[6] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. Mukherjee, and R. Rangan. Computing architectural vulnerability factors for address-based structures. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 532–543, 2005.

[7] 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.

[8] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 83–94, June 2000.

[9] D. Burger and T. Austin. The SimpleScalar Toolset, Version 3.0. http://www.simplescalar.com.

[10] J. Garrett and M. Stan. Active Threshold Compensation Circuit for Improved Performance in Cooled CMOS Systems. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, pages 410–413, May 2001.

[11] M. A. Gomaa and T. N. Vijaykumar. Opportunistic transient-fault detection. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 172–183, 2005.

[12] S. Mukherjee. *Architecture Design for Soft Errors*. Morgan Kaufmann/Elsevier, 2008.

[13] S. Mukherjee, M. Kontz, and S. Reinhardt. Detailed Design and Evaluation of Redundant Multithreading Alternatives. In *International Symposium on Computer Architecture (ISCA)*, pages 99–110, May 2002.

[14] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin. A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, pages 29–40, December 2003.

[15] A. Parashar, S. Gurumurthi, and A. Sivasubramaniam. A Complexity-Effective Approach to ALU Bandwidth Enhancement for Instruction-Level Temporal Redundancy. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 376–386, June 2004.

[16] A. Parashar, S. Gurumurthi, and A. Sivasubramaniam. SlicK: Slice-based Locality Exploitation for Efficient Redundant Multithreading. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 95–105, October 2006.

[17] V. Reddy, S. Parthasarathy, and E. Rotenberg. Understanding Prediction-Based Partial Redundant Threading for Low-Overhead, High-Coverage Fault Tolerance. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 83–94, October 2006.

[18] K. Reick, P. Sanda, S. Swaney, J. Kellington, M. Mack, M. Floyd, and D. Henderson. Fault-Tolerant Design of the IBM Power6 Microprocessor. *IEEE Micro*, 28(2):30–38, March 2008.

[19] S. Reinhardt and S. Mukherjee. Transient Fault Detection via Simultaneous Multithreading. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 25–36, June 2000.

[20] E. Rotenberg. AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors. In *Proceedings of the International Symposium on Fault-Tolerant Computing (FTCS)*, pages 84–91, June 1999.

[21] E. Schuchman and T. Vijaykumar. Rescue: A Microarchitecture for Testability and Defect Tolerance. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 160–171, June 2005.

[22] N. Seifert, P. Slankard, M. Kirsch, B. Narasimham, V. Zia, C. Brookreson, A. Vo, S. Mitra, B. Gill, and J. Maiz. Radiation-Induced Soft Error Rates of Advanced CMOS Bulk Devices. In *Reliability Physics Symposium Proceedings*, March 2006.

[23] 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.

[24] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi. Modeling the Effect of Technology Trends on Soft Error Rate of Combinational Logic. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, June 2002.

[25] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-Aware Microarchitecture. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 1–13, June 2003.

[26] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-Aware Microarchitecture: Extended Discussion and Results. Technical Report CS-2003-08, CS Department, University of Virginia, April 2003.

[27] J. Smolens, B. Gold, J. Kim, B. Falsafi, J. Hoe, and A. Nowatzyk. Fingerprinting: Bounding Soft-Error Detection Latency and Bandwidth. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 224–234, October 2004.

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

[29] J. Srinivasan, S. Adve, P. Bose, and J. Rivers. The Case for Lifetime Reliability-Aware Microprocessors. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 276–287, June 2004.

[30] J. Srinivasan, S. Adve, P. Bose, and J. Rivers. The Impact of Technology Scaling on Lifetime Reliability. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 177–186, June 2004.

[31] J. Srinivasan, S. Adve, P. Bose, and J. Rivers. Exploiting Structural Duplication for Lifetime Reliability Enhancement. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 520–531, June 2005.

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

[33] K. Walcott, G. Humphreys, and S. Gurumurthi. Dynamic Prediction of Architectural Vulnerability from Microarchitectural State. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 516–527, June 2008.

[34] C. Weaver, J. Emer, S. Mukherjee, and S. Reinhardt. Techniques to Reduce the Soft Error Rate of High-Performance Microprocessor. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2004.