Lifetime Reliability Aware Architectural Adaptation

Thannirmalai Somu Muthukaruppan, Tulika Mitra
Department of Computer Science
National University of Singapore
{*tsomu,tulika*}*@comp.nus.edu.sg*

***Abstract*—Relentless CMOS technology scaling has resulted in increased on-chip temperature leading to serious concerns about lifetime reliability of micro-processors. Though dynamic thermal management techniques can control peak temperature, they often fail to meet the reliability targets due to the complex interplay between temperature and reliability. In this paper, we propose a dynamic reliability management (DRM) technique that exploits architectural adaptation in conjunction with dynamic voltage/frequency scaling (DVFS). We employ an online Bayesian classifier that can efficiently detect the reliable configurations, while a performance prediction model selects the one with best performance among all the reliable configurations. We later extend our approach to meet both reliability and thermal constraints. Experimental results reveal that our adaptive DRM technique achieves reliability targets while reducing performance overhead by 42.30% compared to DVFS and 28.68% compared to DVFS with fetch gating.**

I. Introduction

Technological scaling in deep sub-micron level has resulted in increased power density and on chip temperature, which directly impact the processor lifetime reliability. Extensive studies have demonstrated the detrimental impact of non-ideal scaling on permanent errors caused by wear out phenomena such as electro migration, stress migration, gate oxide breakdown and thermal cycles [13]. As the lifetime degradation at future technology generations is expected to increase, it has become important to design reliability solutions at the architectural level.

The goal of any dynamic reliability management (DRM) technique is to improve the lifetime reliability of the microprocessor with minimal impact on performance. Most applications have limited instruction-level parallelism (ILP) and cannot take advantage of extensive ILP exploitation techniques present in current generation out-of-order architectures. When the ILP of an application is limited, various architectural parameters can be scaled down to the appropriate level such that performance remains the same but the power density and consequently the reliability can be improved. In this paper, we propose a dynamic reliability management (DRM) technique that adapts at runtime the following micro-architectural parameters: (a) supply voltage and frequency, (b) fetch gating (fraction of cycles where instruction fetching is disabled), (c) issue width (number of instructions issued per cycle), (d) instruction window size (maximum number of instructions in-flight), and (e) cache way disabling (number of cache ways disabled in the set-associative cache). The detailed rationale behind the choice of these parameters will be presented in Section III.

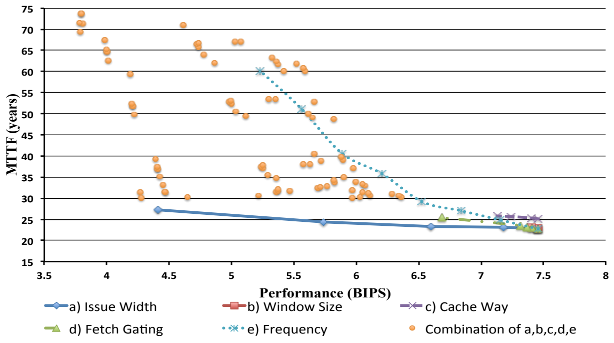

Figure 1: MTTF vs. Performance for different adaptation mechanisms for the benchmark bzip2

Figure 1 shows the impact of different adaptation mechanisms on both mean time to failure (MTTF) and performance corresponding to the benchmark *bzip2*. MTTF is defined as the mean time expected until the first failure of the processor. The experimental setup will be described in Section V. We assume the lifetime reliability target (MTTF) as 30 years. The performance is plotted as billion instructions processed per second (BIPS). The baseline contains the highest performing value for each adaptation parameter and hence provides maximum performance with minimum reliability (7.5 BIPS with MTTF equal to 21 years). Then we adapt each parameter individually, while keeping the remaining parameters constant at the highest performing value. Clearly, voltage/frequency scaling has the largest steps and can improve reliability to more than 60 years at significantly lower performance of 5.25 BIPS. The other architectural mechanisms, in contrast, improve reliability moderately with little impact on performance. The figure also shows that adapting a combination of these parameters (the yellow points in Figure 1) can satisfy the reliability target with much better performance compared to DVFS alone. Our objective, thus, is to engage a combination of these parameters so as to reach the reliability target without sacrificing much performance.

It is challenging to design a DRM technique that exploits multiple different architectural mechanisms in conjunction with DVFS. We need to identify, in each adaptation interval at runtime, the optimal configuration (choice of values for the different parameters) that meets the reliability target with the best performance for an application (or the phase of an application). To filter out unreliable configuration points, we design a software-based Bayesian classifier [15]. In order to identify the optimal configuration among the reliable ones, we develop an analytical model that can predict the performance of a configuration corresponding to the currently executing application.

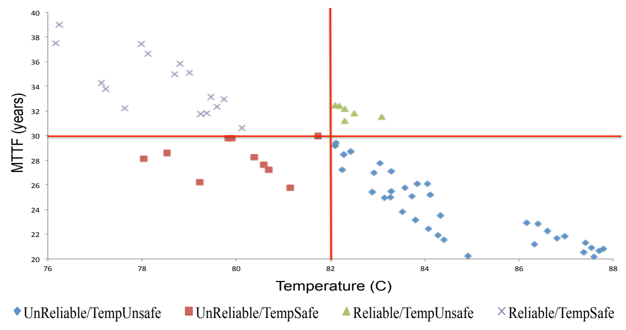

Figure 2: MTTF vs. temperature for different architectural configurations for the benchmark crafty

Due to the exponential dependency of lifetime reliability on the temperature [12], one can expect the dynamic thermal management solutions (DTM) to be employed for DRM. However, there exist subtle differences between temperature and reliability management goals. The objective of DTM techniques is to optimize performance while keeping peak temperature below certain threshold. On the other hand, DRM maximizes performance while meeting lifetime reliability target. Lifetime reliability is impacted through chip-wide higher temperature rather than just the peak temperature at a particular localized structure. Moreover, certain mechanism like DVFS used for thermal management might have negative impact on reliability [12]. The need for independent but synergistic DTM and DRM techniques is illustrated in Figure 2. We plot MTTF versus peak temperature observed for different micro-architectural configurations corresponding to benchmark *crafty*. We assume peak temperature threshold of 82^{o}C and lifetime reliability target (MTTF) as 30 years. We partition the graph into four regions (clockwise): (1) thermally safe and reliable, (2) thermally unsafe and reliable, (3) thermally unsafe and unreliable, and (4) thermally safe and reliable. It is evident from Figure 2 that there exist configurations in all the four partitions. Therefore, it is imperative that we design customized techniques specially targeted at improving reliability.

II. Related Work

Traditionally, DTM techniques were employed as a convenient proxy to improve the lifetime reliability of the processors [12]. Commonly employed mechanisms that reduce temperature include DVFS, activity migration [6], [11], fetch gating and clock gating. However, these techniques do not consider the lifetime reliability problem explicitly.

Several techniques have been proposed for lifetime reliability (also known as hard errors) management. Srinivasan et al. [12] proposed an architectural level analytical model, called Reliability-Aware Micro-Processor (RAMP), for temperature induced lifetime reliability. They explore the effectiveness of optimizing the architectural configurations and the voltage/frequency settings statically to meet the reliability target. Karl et al. [10] proposed the use of a proportional-integral-derivative (PID) controller based DRM technique. The most common technique employed for DRM is DVFS, possibly with a feedback controller. Dynamic wearout centric job scheduling in chip multiprocessor proposed in [7] employs a fine grained reliability management at the module-level of the cores. As these approaches focus only on the lifetime reliability, the peak temperature constraint is not considered.

We show that dynamically adapting architectural configurations along with DVFS can provide better performance and meet both reliability and/or thermal constraints. Also, while previous works are mostly reactive in nature, i.e., the performance is throttled only when reliability constraint is violated, we propose a predictive DRM technique.

III. Parameter Selection

We identify the following parameters as potential adaptation candidates in conjunction with DVFS: (a) fetch gating, (b) issue width, (c) instruction window size, and (d) selective cache way disabling. These parameters are easy to adapt at runtime and also have considerable impact on temperature and lifetime reliability. We choose eight different frequency levels (3.6GHz to 2.5GHz) for DVFS. We use five different fetch gating levels: 0 – 4. When the fetch gating level is set to T ($1 \leq T \leq 4$), the fetch unit disables fetching once every T cycles (0 being the default no fetch gating configuration). We employ five different issue widths: 2–6. When the issue-width is altered, the additional functional units and the appropriate register file ports are disabled so as to reduce leakage power. The instruction window can be scaled to four different sizes: 16, 32, 48, and 64 instructions. The adaptation is achieved by dividing the instruction window into four banks of equal sizes, each containing 16 instructions. Each bank can be enabled or disabled independently [4]. We have to wait for all the instructions from a bank to be committed before it can be disabled. Thus instruction window resizing has more overhead compared to fetch gating and issue width scaling. We assume a 4-way set-associative 64KB L1 data cache. The data cache can be resized through selective cache way disabling [1]. We can thus achieve 16KB to 64KB L1 data cache size. To ensure the correctness of the program, the blocks from the disabled cache ways have to be flushed before they are disabled. Note that we only adapt the data cache and not the

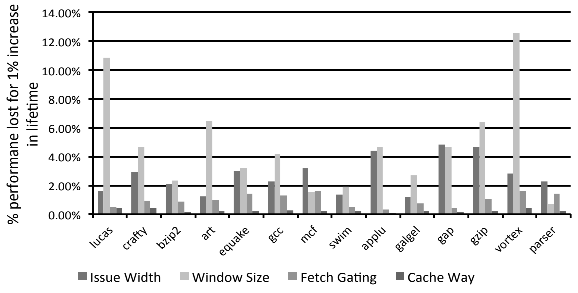

Figure 3: Performance-reliability tradeoff.

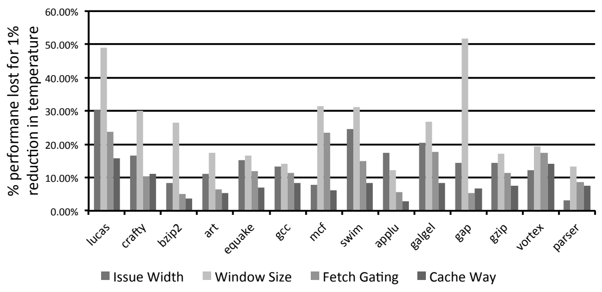

Figure 4: Performance-temperature tradeoff.

instruction cache. This is because fetch gating can achieve similar effect as instruction cache resizing. We assume that the architecture has specific instructions that can change the configurational parameters at runtime.

The best adaptation mechanism is the one that can satisfy the reliability or the thermal targets with minimal impact on performance. For each benchmark program from SPEC 2000, we first identify the most compute intensive phase that leads to either increased steady-state temperature or worst reliability under the baseline non-adaptive configuration (see Section V). Next we adapt each parameter individually and quantify its impact on improving the reliability (or reducing the steady-state temperature) of the identified phase.

Figure 3 (Figure 4) shows the percentage of performance lost in order to increase the lifetime reliability MTTF (reduce the steady-state temperature) by 1% compared to the default configuration. As each parameter has a range of values, we find the mean performance lost for 1% increase in MTTF (or decrease in temperature) compared to the default configuration. It is obvious from the figures that instruction window resizing contributes to serious performance degradation while attempting to improve either reliability or temperature. This is because, scaling window size has only localized impact on power consumption and considerable performance overhead per transition. Therefore, we decide to eliminate window resizing from further consideration.

In terms of reliability, cache way disabling is the clear winner with minimal impact on performance. There are two reasons behind this behavior. First, for applications with smaller memory footprint, small data cache size suffices and reduces power consumption by disabling the unused cache ways. For applications with larger memory footprint, smaller cache size increases the number of cache misses and thus reduces the switching activity in the core due to the delayed delivery of data from memory. This leads to reduced power density in the back end of the core and hence increased reliability. Even though moderate hardware modifications are required for selective cache way disabling [1], the benefits are considerable from both thermal and reliability management perspective. However, employing only cache way disabling is not sufficient to meet the target lifetime reliability. From Figure 1, it is evident a combination of mechanisms should be employed to achieve the desired MTTF. Thus, our final set of parameters adapted at runtime are issue-width scaling, fetch gating, selective data cache way disabling in conjunction with DVFS.

IV. Dynamic Reliability Management

We now present our dynamic reliability and thermal management framework based on architectural adaptation. The reliability and/or thermal management module is periodically invoked once every adaptation interval: 10^7 cycles or 2.8ms at 3.6 GHz. As our focus is on temperature induced lifetime reliability issues and the temperature changes occur very slowly, we set the adaptation interval in the order of milliseconds. At every adaptation interval, we first check if there is any significant variation in the workload characteristics, that is, whether there is a new application or the same application has moved into a new phase [5]. If the workload characteristics change, then we may need to adapt the architectural parameters. This is achieved through two major components: (1) the monitoring module, and (2) the configuration search module.

Monitoring module: The monitoring module employs a combination of measurements and modeling to estimate the MTTF and the temperature corresponding to the current workload. We assume that the processor is fitted with circuit-level multi-use sensors similar to the ones presented in ElastIC architecture [14]. These sensors can characterize performance, lifetime degradation, temperature, and power consumption at fine granularities. Once the data from the sensors and the physical parameters (such as supply voltage, current, and activity factor) are collected, the MTTF and the temperature are estimated. The MTTF estimation relies on the RAMP [12] model and is entirely implemented in software. The MTTF computation in the RAMP model involves complex operations and frequent exponentiation, which are avoided through pre-computation and fast exponentiation. Thus the overhead is estimating MTTF and temperature is negligible compared to the adaptation interval.

Configuration Search Module: The goal of this module is to select the configuration with maximum performance that satisfies the MTTF and/or thermal constraints. The configuration search module is also implemented entirely in software and consists of two major components: A) naive Bayesian classifier and B) performance prediction module.

A. Naive Bayesian Classifier

Our objective in configuration search is to quickly filter out the unreliable and/or thermally unsafe configurations. We employ a naive Bayesian classifier for this purpose. Classification problems are characterized by the need to classify an input pattern into one of the output categories. Among the various classifiers (naive Bayesian, decision trees and neural network) available, we chose naive Bayesian classifier [15] because its simplicity allows each input pattern to contribute towards the final classification decision. It offers several additional advantages such as fast training time, minimal computation time, and the ability to add new attribute without re-training.

For our configuration filtering problem, we need to select both the workload characteristics and the adaptive architectural configurations as input parameters. Thus each input pattern consists of seven parameters: (a) issue width, (b) fetch gating level, (c) operating frequency, (d) number of integer instructions issued per cycle, (e) number of floating point instructions issued per cycle, (f) number of memory instructions issued per cycle, and (g) number of branch instructions issued per cycle. The output is yes or no classification indicating whether the workload, configuration pair satisfy the reliability and/or the thermal constraint. Note that the inputs to the classifier are the number of instructions issued rather than number of instructions committed because the number of instructions issued influence the temperature and hence the MTTF of the microprocessor, whereas the number of instructions committed determines the performance of the microprocessor. This estimation of number of instructions issued per cycle is obtained through the performance prediction model discussed in Section IV-B. Among the architectural parameters, we decide to leave out the number of cache ways as an input to the classifier. This is because the effect of cache resizing can be captured sufficiently with the performance prediction model.

We train the classifier off-line either during system installation and/or when the system conditions (e.g., ambient temperature) change. The training set is generated by running a set of micro-benchmarks under various configurations and checking if the MTTF and/or thermal constraints are satisfied. The micro-benchmarks contain loops with varying mix of integer, floating point, memory, and branch instructions. The instruction mix are generated in a pseudo random fashion to account for the variability in the workloads that may execute on the processor. To overcome the problem of random sampling, we employ Latin hypercube sampling to enumerate 100 representative configurations from the configuration space. We train the classifier with 50 micro-benchmarks each running on the 100 selected configurations.

After training, we test our classifier using a number of SPEC 2000 benchmarks. We simulate each benchmark at 100 configurations points and determine if the execution violates the MTTF constraint. We compare our simulation outcome with the corresponding output from the classifier. Classification errors can be categorized into false positive and false negative. A classifier commits false positive error when it erroneously classifies a reliable configuration point as unreliable. False negative errors are committed when the classifier erroneously classifies an unreliable configuration point as reliable. We observe that our classifier is very accurate with only 6.4% false negatives and 8.5% false positives, on an average, across all the benchmarks. Note that the only impact of a false positive error is reduced performance as the configuration will not be selected, while false negative errors may violate the reliability constraint. When a selected configuration fails to meet the reliability target during execution, it will be detected in the monitoring module. The module will then invoke emergency fail safe mechanisms such as clock gating and/or power gating to bring the situation under control. We extend our classifier to incorporate both the MTTF and the temperature constraints to develop our dynamic thermal and reliability management (DTRM) technique. We obtain the training set by running the micro-benchmarks under various configurations and checking if both the MTTF and the temperature constraints are satisfied. We train this combined classifier and observe 7.9% false negative and 9.2% false positive.

B. Performance Prediction Model

The performance prediction model is required for two reasons. First, we need to predict the performance of the reliable and thermally safe configurations for the current workload so as to choose the optimal one. Secondly, the classifier requires the workload characteristics (instruction mix issued per cycle) for a configuration to be classified.

The inputs to the performance prediction model are the number of integer, floating point, memory and branch instructions committed in the previous adaptation interval as well as the total number of instructions committed N_{useful}, which could be obtained from hardware performance counters present in modern microprocessors.

Our performance prediction model is inspired by the interval based models proposed in [8], [9]. The interval based model suggests that there exists a sustained background performance level that is punctuated by transient miss-events such as branch mis-prediction and cache misses. The cycles per instruction (CPI) can be expressed as

$$CPI = CPI_{steady} + CPI_{miss} \tag{1}$$

$$CPI_{miss} = CPI_{bmiss} + CPI_{icmiss} + CPI_{dcmiss} \tag{2}$$

where CPI_{steady} is the sustained background performance in the absence of miss-events and $CPI_{bmiss}, CPI_{icmiss}$ and CPI_{dcmiss} denote the performance loss incurred due to branch mis-predictions, instruction cache miss and data cache miss, respectively. CPI_{miss} can be computed by counting the number of corresponding miss-events and miss penalties as follows,

$$CPI_{miss} = \frac{N_{icmiss} \times P_{icmiss} + N_{bmiss} \times P_{bmiss} + N_{dcmiss} \times P_{dcmiss}}{N_{useful}} \quad (3)$$

where N_{icmiss}, N_{bmiss} and N_{dcmiss} are the number of instruction cache miss, data cache miss, and branch mispredictions over our adaptation interval. Data cache miss can be further divided into L1 data cache miss $CPI_{d1cmiss}$ and L2 cache miss $CPI_{d2cmiss}$. The penalty values (P_{icmiss}, P_{bmiss} and P_{dcmiss}) are computed using the first order superscalar model [9]. We observe that the performance impact of miss events $CPI_{bmiss}, CPI_{icmiss}$, $CPI_{d2cmiss}$ are fairly constant across configurations except for L1 data cache ($CPI_{d1cmiss}$), which we adapt. This is because changing issue width and fetch gating has minimal impact on the number of miss events.

Memory Exploration Module: As we adapt L1 data cache dynamically, $CPI_{d1cmiss}$ varies across the cache configurations. When a new program phase $P^{'}$ is encountered, the memory exploration module is triggered. As there are only four data cache configurations, we execute phase P' with all the four cache configurations, one per adaptation interval. We sample and memoize the L1 data cache miss rate $CPI_{d1cmiss}$ for the various cache configurations. We use this information if phase $P^{'}$ is encountered again.

Prediction: For the current configuration **C**,

$$CPI_{steady}(C) = CPI(C) - CPI_{miss}(C) \quad (4)$$

$$CPI_{miss}(C) = CPI^{'}_{miss}(C) + CPI_{d1cmiss}(C) \quad (5)$$

where $CPI^{'}_{miss}(C)$ represents all the miss events except L1 data cache miss. In general, at a new configuration $C^{'}$

$$CPI^{'}_{miss}(C^{'}) = CPI^{'}_{miss}(C) \quad (6)$$

Now the CPI for the configuration $C^{'}$ can be expressed as,

$$CPI(C^{'}) = CPI_{steady}(C^{'}) + CPI^{'}_{miss}(C) + CPI_{d1cmiss}(C^{'}) \quad (7)$$

$CPI_{d1cmiss}$ is provided by the memory exploration module.

We now need to determine $CPI_{steady}(C^{'})$. At fetch gating level T, the number of instructions delivered per cycle to the pipeline is $\frac{T}{T+1} \times FW$, where FW is the fetch width. In the steady state, the number of instructions issued per cycle must be the same as the number of instructions fetched per cycle. Thus $IPC_{steady}(C^{'})$ at configuration $C^{'} = \langle IW, FG \rangle$ can be expressed as

$$IPC^{ideal}_{steady}(IW, FG) = min(IW, \frac{T}{T+1} \times FW) \quad (8)$$

We refer $IPC_{steady}(C^{'})$ as $IPC^{ideal}_{steady}(C^{'})$ because the equation assumes that all the instructions are of unit latency. To account for variable functional unit latency, we compute a ratio μ between ideal steady state IPC and observed steady state IPC for the current configuration C.

$$\mu = \frac{IPC_{steady}(C)}{IPC^{ideal}_{steady}(C)} = \frac{1}{CPI_{steady}(C) \times IPC^{ideal}_{steady}(C)} \quad (9)$$

As the latency of the functional units are not adapted, the value μ remains constant across the configuration.

$$IPC_{steady}(C^{'}) = \mu \times IPC^{ideal}_{steady}(C^{'}) \quad (10)$$

$$CPI(C^{'}) = \frac{1}{IPC_{steady}(C^{'})} + CPI^{'}_{miss}(C) + CPI_{d1cmiss}(C^{'}) \quad (11)$$

Our Bayesian classifier accepts the number of instructions issued corresponding to different classes as input.

$$IPC^{X}_{issue}(C^{'}) = IPC_{issue}(C^{'}) \times \frac{N^{X}_{useful}}{N_{useful}} \quad (12)$$

where $IPC^{X}_{issue}(C^{'})$ (N^{X}_{useful}) is the number of instructions issued per cycle (total number of instructions committed) of type X (integer, floating point, memory, or branch).

C. Search Space Pruning

Our configuration design space consists of four axes (frequency, issue width, fetch gating and cache ways). From Equation 8, it is evident that IPC_{steady} is limited by both issue width and fetch gating. Increasing either of them alone will not facilitate increase in performance. Thus the four dimensional design space can be reduced to three dimensional design space (frequency, cache ways, and IPC_{steady}). For each point in this space, the classifier determines whether that particular configuration meets the constraints. The search process can be further optimized by doing linear search for frequency, cache ways and binary search along the IPC_{steady} axis. This optimization is based on the fact that at a particular frequency level $F^{'}$ and steady IPC $IPC^{'}$, if $IPC^{'}$ does not meet the MTTF constraints, then all the configurations with IPC_{steady} values higher than $IPC^{'}$ will not meet the reliability or temperature constraint either. This is because higher performance leads to higher temperature, which has negative impact on the lifetime reliability. For F frequency levels, I levels of IPC_{steady} values and C cache configurations, the maximum number of configurations explored are $O(F \times C \times ln(I))$. In our adaptive framework, we have eight frequency values, six IPC steady values and four cache ways, resulting in a total of 192 configuration points. The configuration search module takes approximately 10K cycles (2.8 μs at 3.6 Ghz) to determine the optimal point in the worst case.

V. Experimental Evaluation

We use SimpleScalar [2] simulator with Wattch [3] power models for our experiments. Our baseline non-adaptive processor is modeled as 6-way issue, 64-entry instruction window, 64 KB L1 data and instruction cache, 2MB L2 unified cache and 4K entry bimod branch predictor. Our adaptive architecture has four possible issue widths (2–6), five possible fetch gating (1–4, no fetch gating), four possible L1 data cache sizes (64KB – 16KB). We vary the processor frequency from 3.6 GHz to 2.5 GHz. We assume 10 μs penalty to change the frequency settings [16]. We use Hotspot-5.0 [11] for thermal simulation with a floor plan similar to Alpha 21364. To include the effects of temperature on leakage power, we use the leakage power density value provided in [13]. RAMP [12] is employed to evaluate the lifetime reliability. We set the reliability budget (MTTF) in our experiments as 30 years [12]. For DTRM technique, the maximum temperature allowed is 82^{o}C. We use 14 benchamarks from SPEC 2000.

We compare our architectural adaptation based DRM technique, called **Adaptive**, with the following approaches: (a) **DVFS:** state-of-the-art hardware based DRM technique employing only DVFS. We use a PI-controller based scheme [6], and (b) **Freq-FG:** a technique combining two different mechanisms for reliability management, PI-controlled DVFS and fetch gating [16]. For mild stress, a constant fetch gating level of 3 is engaged. As the stress becomes severe, controller based DVFS is employed.

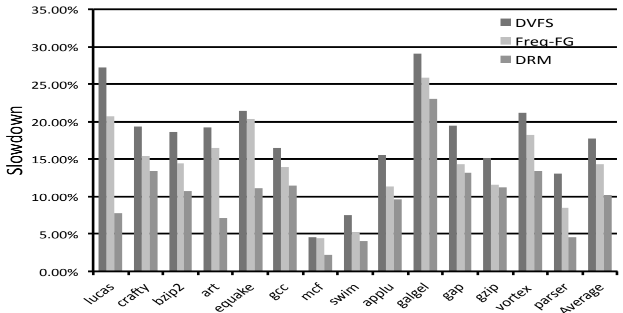

Figure 5: Comparison of different DRM techniques

Figure 5 plots the slowdown in performance compared to the baseline non-adaptive architecture at maximum frequency (without any thermal or reliability constraints). Our adaptive technique outperforms others, i.e., it achieves lower performance degradation. On an average, *Adaptive* has 10.22% slowdown, while *DVFS* and *Freq-FG* have 17.72% and 14.33% slowdown, respectively. Thus *Adaptive* reduces performance degradation by 42.30% compared to *DVFS* and 28.68% compared to *Freq-FG*.

Figure 6 plots the time varying trends in IPC, frequency, architectural parameters, and performance (BIPS) for *bzip2*. These plots provide insight into why *Adaptive* performs better. A higher value of an architectural parameter implies better performance. We do not adapt the architectural parameters in *Base* and *DVFS*. *Adaptive* manages to operate at higher frequency (and thus have better performance) because it scales micro-architecture structures to reduce power consumption. We also observe more transitions in frequency in other techniques compared to *Adaptive*, resulting in thermal cycling and consequently worse reliability.

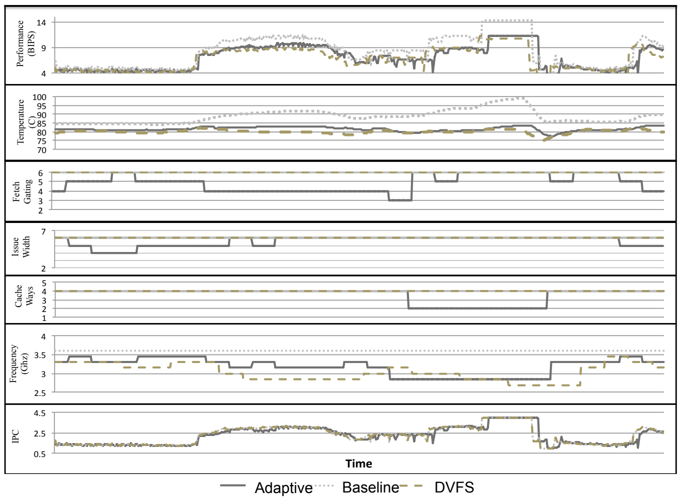

Figure 6: Time varying trends for bzip2.

As there is no existing techniques for integrated temperature and reliability management, we compare our DTRM technique with the DRM technique. The set point for DRM technique is only MTTF=30 years but DTRM technique has the additional set ppint for temperature (82^oC). DTRM technique is an easy extension of our adaptive DRM technique where the classifier is trained to choose the configurations that meet both the temperature and the reliability target. We observed that, the DTRM technique, on an average, has to sacrifice 11.95% performance to meet both the temperature and the reliability targets.

VI. Conclusions

We propose a dynamic reliability management technique that adapts micro-architectural parameters in conjunction with DVFS. Our adaptive method achieves the reliability target while reducing performance overhead by 42.30% compared to DVFS alone and 28.68% compared to DVFS with fetch gating. We also extend our technique to incorporate temperature constraints along with reliability constraints.

Acknowledgment

This work was supported by Singapore Ministry of Education Academic Research Fund Tier 2, MOE2009-T2-1-033.

References

[1] D.H. Albonesi. Selective cache ways: on-demand cache resource allocation. In *MICRO, 1999.*

[2] Austin et al. Simplescalar: an infrastructure for computer system modeling. *IEEE Computer*, 2002.

[3] Brooks et al. Wattch: a framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News*, 2000.

[4] Buyuktosunoglu et al. A circuit level implementation of an adaptive issue queue for power-aware microprocessors. GLSVLSI, 2001.

[5] Dhodapkar et al. Managing multi-configuration hardware via dynamic working set analysis. In *ISCA, 2002.*

[6] Donald et al. Techniques for multicore thermal management: Classification and new exploration. ISCA, 2006.

[7] Feng et al. Maestro: Orchestrating lifetime reliability in chip multiprocessors. In *High Performance Embedded Architectures and Compilers*. 2010.

[8] Jayaseelan et al. Dynamic thermal management via architectural adaptation. In *DAC*, 2009.

[9] Karkhanis et al. A first-order superscalar processor model. In *ISCA, 2004.*

[10] Karl et al. Reliability modeling and management in dynamic microprocessor-based systems. DAC, 2006.

[11] Skadron et al. Temperature-aware microarchitecture. *SIGARCH Comput. Archit. News*, 2003.

[12] Srinivasan et al. The case for lifetime reliability-aware microprocessors. *SIGARCH Comput. Archit. News*, 2004.

[13] Srinivasan et al. The impact of technology scaling on lifetime reliability. DSN, 2004.

[14] Sylvester et al. Elastic: An adaptive self-healing architecture for unpredictable silicon. *Design Test of Computers, IEEE*, 2006.

[15] Zhao et al. Face recognition: A literature survey. *ACM Comput. Surv.*, 2003.

[16] Kevin Skadron. Hybrid architectural dynamic thermal management. DATE, 2004.