Facelift: Hiding and Slowing Down Aging in Multicores*

Abhishek Tiwari and **Josep Torrellas**
Department of Computer Science
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

Abstract

Processors progressively age during their service life due to normal workload activity. Such aging results in gradually slower circuits. Anticipating this fact, designers add timing guardbands to processors, so that processors last for a number of years. As a result, aging has important design and cost implications.

To address this problem, this paper shows how to hide the effects of aging and how to slow it down. Our framework is called Facelift*. It hides aging through aging-driven application scheduling. It slows down aging by applying voltage changes at key times — it uses a non-linear optimization algorithm to carefully balance the impact of voltage changes on the aging rate and on the critical path delays. Moreover, Facelift can gainfully configure the chip for a short service life. Simulation results indicate that Facelift leads to more cost-effective multicores. We can take a multicore designed for a 7-year service life and, by hiding and slowing down aging, enable it to run, on average, at a 14–15% higher frequency during its whole service life. Alternatively, we can design the multicore for a 5 to 7-month service life and still use it for 7 years.*

1 Introduction

The challenges of ensuring the reliability of upcoming, deep sub-micron hardware have spurred interest in the fact that processors age or wear-out while executing their normal workloads [7]. In particular, the maximum frequency that a processor can deliver decreases slowly and gradually with time [4].

Two major mechanisms that induce progressive slowdown in processors are Negative Bias Temperature Instability (NBTI) and Hot-Carrier Injection (HCI) [4]. Roughly speaking, these effects are due to stresses induced on transistors by the normal, continuous movement of charges. At the macroscopic level, these effects manifest as gradually slower transistors and, hence, gradually slower critical paths.

Anticipating this fact, processor designers add timing guardbands to their designs [3]. The goal of guardbands is to absorb any increase in critical path delay during the processor's service life, and avert any timing error. Anecdotal evidence suggests that current processors include a guardband to last for 7–10 years. Clearly, the aging process has important implications on the design and cost of processors.

While aging or wearout has been extensively studied at the device level, there is relatively little work at the architecture or system level. Specifically, Srinivasan *et al.* [29, 30] focus on modeling the Mean Time To Failure (MTTF) due to aging mechanisms. The value of such MTTF is typically multiple times the expected service life of a processor, since one should expect that only a negligible fraction of processors will fail during their service life. The authors propose voltage, frequency, and microarchitectural adaptations to attain the required MTTF more cost-effectively.

Our goal is to understand how critical path delays increase due to aging *during the service life* of the processor, and then use inexpensive techniques to reduce the performance degradation. The most relevant work that we are aware of is that of Ramakrishnan *et al.* [22], Abella *et al.* [1], and Shin *et al.* [26]. Their idea is to dynamically set the transistors to a logic value that "undoes" some of the aging process — during periods when this does not disrupt processor execution. While their techniques are effective and transparent to software layers, if they are applied widely across processor structures, they are likely to be intrusive to the processor design and may have performance implications. Ideally, we would like approaches that do not affect processor internals.

In practice, aging depends exponentially on high-level parameters that can be easily manipulated — supply voltage (V_{dd}), temperature (*T*), and threshold voltage (V_t). Small changes to these parameters at key times in the processor's service life can have major effects without requiring intrusive designs.

Using this general approach, this paper contributes with a framework of techniques to (i) hide the effects of aging in a multicore, (ii) slow down aging, and (iii) gainfully configure the chip for a short service life. We call our framework *Facelift*. A second contribution is to show how the shorter guardband enabled by Facelift can be used to either (i) design a less refined version of the processor or (ii) clock the processor at a higher frequency.

Facelift hides the effects of aging in a multicore by steering high-*T* jobs to the fast cores and low-*T* ones to the slow cores. Keeping the slow cores cooler enables the chip to appear to age less. Facelift slows down aging by making small, chip-wide changes to V_{dd} or V_t at key times — using a non-linear optimization algorithm to carefully balance the impact of the changes on the aging rate and on the critical path delays. Finally, Facelift configures a chip for a short service life by "shifting" performance from the unused lifetime portion to the used one.

Simulation results indicate that the Facelift techniques lead to more cost-effective multicore designs. We can take a multicore designed for a 7-year service life and, by hiding and slowing down aging, enable it to run, on average, at a 14–15% higher frequency during its whole service life. Alternatively, we can design the multicore for a 5 to 7-month service life and still use it for 7 years. Finally, the implementation of the Facelift techniques is very simple.

This paper is organized as follows. Section 2 provides a back-

*This work was supported by the National Science Foundation under grant CPA-0702501 and by SRC GRC under grant 2007-HJ-1592. Abhishek Tiwari is now with Goldman Sachs, New York City.

ground; Section 3 discusses the impact of aging on architecture; Section 4 presents the techniques to hide and slow down aging, and configure for a short service life; Sections 5 and 6 evaluate Facelift; and Section 7 discusses related work.

2 Background

During a processor's normal, failure-free use, semiconductor-level mechanisms gradually cause devices to become slower. At the macroscopic level, this results in critical paths in the processor gradually and slowly taking longer — an important part of a process that is popularly known as wearout or aging [4]. To tackle this effect, processor designers add timing guardbands to their designs [3], so that any increased critical path delays during a processor's expected service life can be absorbed by the guardband. Informal observations indicate that current processors include a guardband to last for 7–10 years, and possibly less for mobile devices.

According to Bernstein *et al.* [4], the two key mechanisms that increase the delay of transistors during their normal, failure-free operation are Negative Bias Temperature Instability (NBTI) and Hot-Carrier Injection (HCI). In particular, NBTI is a dominant effect that has been the subject of much interest (e.g., [10, 15, 19, 21, 36]). An important insight is that both NBTI and HCI cause a gradual elevation of the threshold voltage (V_t) of transistors [4] — PMOS transistors in NBTI and NMOS transistors in HCI. A higher V_t in turn increases the transistor switching delay (T_s) through the alpha power law [24], where $\alpha \approx 1.3$:

$$T_s \propto \frac{V_{dd} L_{eff}}{\mu (V_{dd} - V_t)^{\alpha}} \quad (1)$$

To propose architectural mechanisms to hide or slow down aging, we need to understand what factors directly impact the increase in transistor delay due to NBTI and HCI. We do this in Sections 2.1 and 2.2. The formulas in these sections correspond to 32nm technology. Bernstein *et al.* [4] also indicate that electromigration in wires is another important mechanism observed during aging. However, since we are unaware of any models in the public domain that suggest how wire delays are affected by electromigration during normal, failure-free operation, we neglect wire delay changes. Finally, Section 2.3 discusses the related issue of process variation.

2.1 NBTI

NBTI is explained by the Reaction-Diffusion model [20]. When logic input 0 is applied to the gate of a PMOS transistor (V_{gs}=$-V_{dd}$), the presence of holes in the channel causes Si-H bonds to break at the interface between the gate oxide and the channel. The resulting H diffuses away, leaving positive traps (Si^+) in the interface, which increase V_t [15]. This process is called the *Stress* phase. The reaction rate mainly depends on the temperature (*T*) and the supply voltage (V_{dd}). The increase in V_t is [36]:

$$\Delta V_{t\_stress} = A_{NBTI} \times t_{ox} \times \sqrt{C_{ox}(V_{dd} - V_t)} \times e^{(\frac{V_{dd}-V_t}{t_{ox} E_0} - \frac{E_a}{kT})} \times t_{stress}^{0.25} \quad (2)$$

where t_{stress} is the time under stress, t_{ox} is the oxide thickness ($0.65nm$), and C_{ox} is the gate capacitance per unit area ($4.6 \times 10^{-20} F/nm^2$). E_0, E_a, and *k* are constants equal to 0.2 V/nm, 0.13 eV, and $8.6174 \times 10^{-5} eV/K$, respectively. A_{NBTI} is a constant that depends on the aging rate.

When logic input 1 is applied to the gate ($V_{gs} = 0$), the transistor turns off, and H atoms diffuse back, eliminating some of the traps. This process is called the *Recovery* phase. The final increase of V_t after considering both the stress and recovery phases is [36]:

$$\Delta V_t = \Delta V_{t\_stress} \times (1 - \sqrt{\eta \times t_{rec}/(t_{stress} + t_{rec})}) \quad (3)$$

where t_{rec} is the time under recovery and η is a constant equal to 0.35.

2.2 HCI

In HCI, electrons accelerated in the electric field of the channel collide with the gate oxide interface. The collision creates electron-hole pairs. Energetic electrons — referred to as "hot" — get trapped in the gate oxide layer, causing an increase in V_t [31]. Hot-carrier induced degradation mainly affects NMOS transistors. Moreover, since hot electrons are generated during logic transitions, the impact of HCI is directly proportional to the switching frequency. The increase in V_t with time is empirically found to follow a power law relationship [31]:

$$\Delta V_t = A_{HCI} \times \alpha \times f \times e^{\frac{V_{dd}-V_t}{t_{ox} E_1}} \times t^{0.5} \quad (4)$$

where *t* is time, and α and *f* are the activity factor and the frequency, respectively. t_{ox} is the oxide thickness ($0.65nm$), and E_1 is a constant equal to 0.8 V/nm [38]. A_{HCI} is a constant that depends on the aging rate.

Finally, the rate of HCI-induced aging also depends on *T*. While, to our knowledge, there is no closed-form analytical relationship between the two, experimental results from [39] show that V_t has a piecewise linear relationship with *T*.

2.3 Process Variation

Process variation is the deviation of transistor parameters from their nominal specification [4]. It is caused by the increasing difficulty to precisely control the fabrication process as feature sizes decrease. Process variation can be die-to-die (D2D) or within-die (WID). WID variation is typically modeled as the combination of a systematic and a random component. The former is induced by limitations of the lithography and other manufacturing processes. Systematic variation exhibits a spatial correlation, meaning that nearby transistors share similar systematic parameter values. On the other hand, random variation is mostly induced by materials effects such as changes in dopant density. Random variation has a different profile for each transistor.

As per Equation 1, variations in transistor parameters such as V_t and gate length (L_{eff}) induce variations in transistor switching delay. At the microarchitecture level, this translates into variations in the delay of logic paths in a processor — as modeled by several microarchitecture models (e.g., [13, 17, 18, 25]). In particular, due to the systematic component of variation, some regions of a chip are slower than others. In a multicore chip, this results in some cores being slower than others. For example, at 32nm, it is estimated that the difference in frequency between the cores in a 20-core chip may reach over 20% [32].

3 Impact of Aging on Architecture

The equations in Section 2 are, to the best of our knowledge, the most accurate descriptions of NBTI and HCI available. They have been validated by experts [15, 20, 31, 34, 36, 39]. Based on them, this section builds a simple model of how aging affects

critical paths in a processor. First, however, we summarize the factors that induce aging.

3.1 Factors that Induce Aging

From Equations 2 and 4, we see that ΔV_t due to NBTI and HCI follows a *power law* with time ($\Delta V_{t\_NBTI} \propto t^{0.25}$ and $\Delta V_{t\_HCI} \propto t^{0.5}$). Since the exponents of *t* are less than one, ΔV_t increases rapidly first and then more slowly — as shown in Figure 1. In the case of NBTI, as given by Equation 3, the increase occurs only while the transistor is under stress, and the recovery phase brings ΔV_t down at a lower rate than it went up.

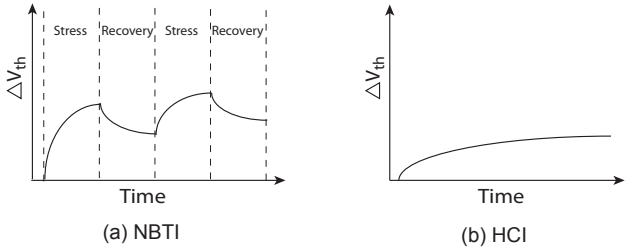


Figure 1: Shape of the change in V_t due to NBTI (a) and HCI (b).

From Equations 2 and 4, we note that ΔV_t increases exponentially with $(V_{dd} - V_t)$ in both NBTI and HCI aging. Moreover, ΔV_t also increases rapidly with *T* — exponentially in NBTI and linearly in HCI. Finally, ΔV_t in HCI also depends linearly on the activity factor α and the frequency *f*. These dependences are summarized in Table 1.

Factor	Impact	
	NBTI	HCI
$V_{dd} - V_t$	exponential	exponential
T	exponential	linear
α, f	—	linear

Table 1: Impact of key factors on aging.

NBTI-induced aging also depends on the duration of the period under stress. What matters is the fraction of the time that the PMOS transistor is stressed, rather than the timing of the interleaving of the stress and recovery periods [16].

3.2 Modeling the Impact of Aging on Critical Paths

We estimate the microarchitectural impact of aging by modeling its effect on a processor's critical paths. Consider first a single transistor. Its switching delay T_s is computed using Equation 1, where $V_t = V_{t0} + \Delta V_t$, and ΔV_t is taken from Equations 3 or 4 depending on whether the transistor is PMOS or NMOS, respectively. Next, we model critical paths in logic structures and in memory structures.

A simple model of a critical path in a logic structure is a chain of FO4 inverters. As shown in Figure 2(a), each CMOS inverter has one transistor of each type. When we change the value of the input of the chain, each inverter relies on one transistor to charge or discharge its output. For example, in Figure 2(b), transistor *T2* discharges node *A*. These transistors are, successively, of different types — in the figure, N, P, N.

The speed at which the signal propagates along the critical path depends primarily on the speed of the transistors that charge or discharge the output nodes. In reality, the other transistors have some effect, but we neglect it. Overall, therefore, we estimate the

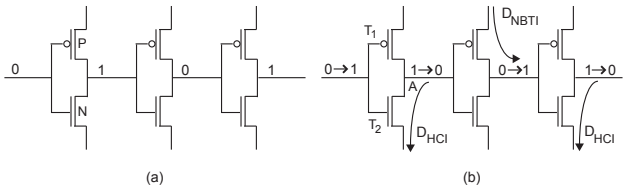


Figure 2: Critical path in a logic structure.

delay of an N-FO4 critical path to be the delay of N/2 NBTI-aged PMOS transistors plus N/2 HCI-aged NMOS transistors.

In reality, a critical path can have a completely different structure and a different ratio of PMOS to NMOS transistors. In this case, we need to identify the number of transistors of each type that need to be activated for the signal to propagate. Then, to compute the critical path delay, we multiply the two curves in Figure 1 by the number of activated transistors of the corresponding type, and add up the results.

In all cases, the curve resulting from combining the NBTI and HCI curves for the individual transistors has the general shape of a power law with an exponent less than one. As a result, it will increase quickly first and then flatten out toward the end.

We model a critical path in a memory structure to be the decoder of the structure, wordline, pass transistor, bitline, and sense amplifier. We use CACTI [33] to estimate the number of cycles taken by the critical path without aging, and the fraction of the time taken by each component. To add the effect of aging, we again use the NBTI and HCI aging of the transistors present in the path, and assume no aging for the wires.

Finally, the NBTI and HCI effects are such that similar nominal devices under similar conditions may age at slightly different rates. Consequently, the aging equations 3 and 4 have a statistical component. This statistical component has been studied only very recently [14], and there is no accepted model. Moreover, it appears that this component is randomly distributed, since its effect comes from the finite number of Si-H bonds in the transistor channel. As a result, its effect tends to average out over the several transistors of a critical path. For these two reasons, we neglect this component in our analysis. Any uncertainty induced by this component will be included in the guardbands selected by the designer.

4 Hiding, Slowing Down, and Consolidating Aging in Multicores

We now propose how to limit the effect of aging. First, we present the Facelift framework to understand the benefits of limiting aging. Then, we examine how to hide aging, slow it down, and configure a processor for a shorter service life.

4.1 Facelift Framework

Consider a processor that, when it is first used, has a critical-path delay equal to C_0 (Figure 3(a)). As the processor is used, the critical path slows down due to aging. After a period *t*=Y_0 equal to the service life for which the processor was designed (e.g., 7 years), aging has caused the critical path delay to reach τ_0. Consequently, for the processor to be usable for the duration of its expected service life, it needs to be clocked at a frequency no higher than $f_0 = 1/\tau_0$. Moreover, the designers need to add a guardband $G_0 = \tau_0 - C_0$ that is gradually consumed (Figure 3(a)).

The same effect is shown in a different way in Figure 3(b). In the figure, the guardband is given as a fraction S_0 of C_0.

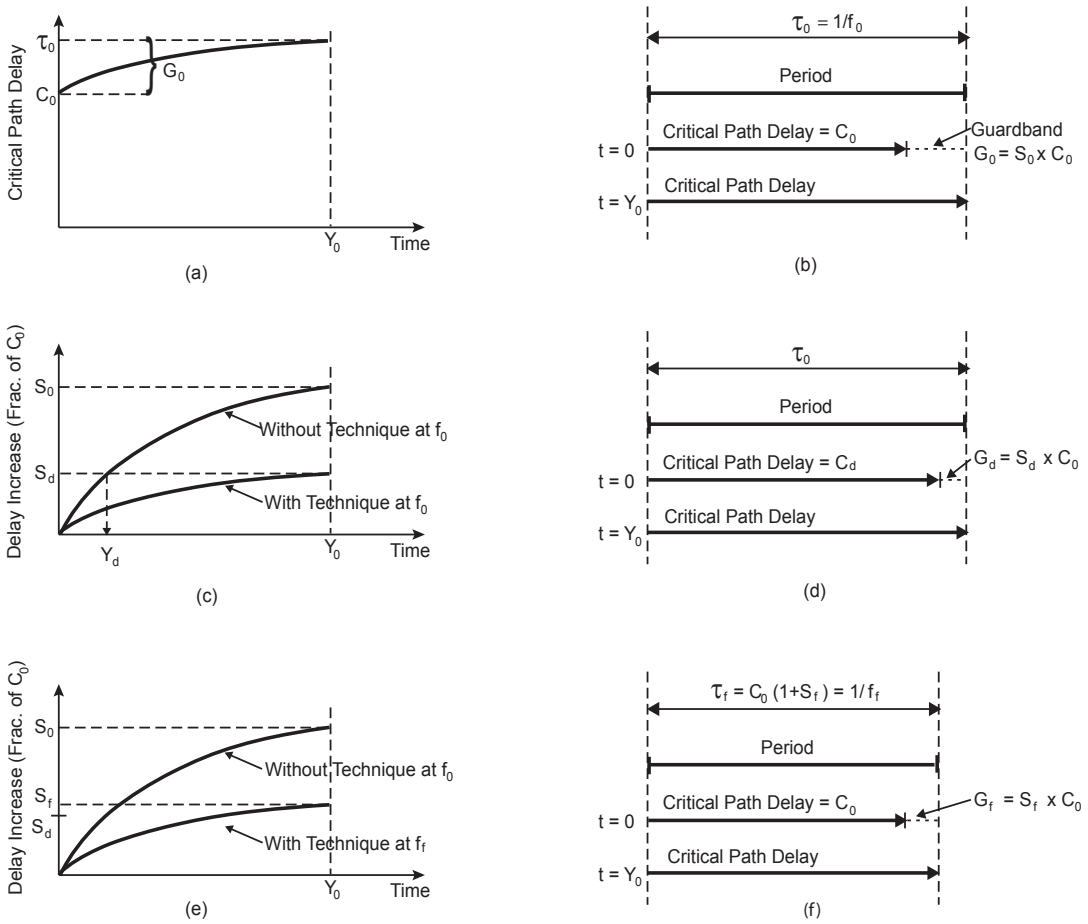


Figure 3: General framework to understand the benefits of limiting aging.

Suppose that we now augment the processor with a technique that limits aging. Figure 3(c) shows the increase in critical path delay as a fraction of C_0 without the technique (upper curve) and with it (lower curve). The upper curve comes from Figure 3(a) with a different Y-axis; by *t*=Y_0, it reaches S_0. By *t*=Y_0, the lower curve reaches only S_d (where *d* stands for *design simplification* approach, as we will see). We will show later that, for $S_0 \approx 0.25$, we attain $S_d \approx 0.09$–0.14.

The reduction in needed guardband can be used in one of two ways: (1) to design a less refined or more "sloppy" version of the processor and still cycle it at the same frequency f_0, or (2) keep the same processor design and cycle it at a higher frequency. We now consider each of these two cases.

4.1.1 Less Refined Design

If designers know that the processor will only age by S_d, they may choose not to speed-tune the paths so carefully (which may save design time), use less sophisticated circuit designs (which may save power), or use cheaper technology or design methodologies. In essence, they can design the processor for a shorter target service life than the standard one.

Figure 3(d) shows the critical path timing in this case. The processor still has a period equal to τ_0. However, at *t*=0, the delay of the critical path is C_d (for targeting *design simplification*), which is longer than C_0 — leaving a guardband $G_d = S_d \times C_0$, which is smaller than G_0. We call C_d/C_0 the *Simplification* factor because it may suggest how much easier it is to design the processor. Observed values from our evaluation are 1.10–1.16, where higher values mean easier designs. After the processor has been in use for the usual *t*=Y_0, the critical path takes no longer than τ_0. This is despite the fact that the processor has been designed with a guardband G_d, which would correspond to an expected service life of only Y_d without the aging-limiting technique (Figure 3(c)).

4.1.2 Higher Frequency

One can instead keep the same processor design and cycle it at a higher frequency f_f (for targeting *frequency*). Note that we do not require the transistors to propagate signals any faster — at *t*=0, the delay of the critical path is C_0, the same as without the technique. We simply live with a smaller guardband. We can do so because the processor ages less. We *do not* change V_{dd}.

Increasing *f* accelerates aging — directly, it increases HCI effects and, indirectly, by boosting *T*, it increases NBTI and HCI effects. As shown in Figure 3(e), the aging-induced increase in delay at Y_0 as a fraction of C_0 is S_f, which is slightly longer than S_d. As a result, we need a guardband $G_f = S_f \times C_0$.

The resulting timing is shown in Figure 3(f). The processor period is a short $\tau_f = C_0 \times (1 + S_f)$. The delay of the critical path at *t*=0 is C_0, like in the case without the technique. Due to the *f* increase, this leaves only a small guardband $G_f = S_f \times C_0$. After the processor has been in use for the usual *t*=Y_0, the critical path delay is no higher than τ_f — even though the processor has been cycling at a higher *f*.

4.1.3 Other Approaches

In the two approaches described, we keep the frequency of the processor constant throughout its service life — f_0 and f_f, respectively. A different way to leverage an aging-limiting technique would be to start the processor's life at a high frequency and gradually decrease it as the processor ages and consumes the guardband. In this paper, we do not consider this approach.

4.2 Hiding Aging: Aging-Driven Scheduling

4.2.1 Description of the Technique

Already in current technologies, multicores experience significant within-die process variation [4]. As a result, the processors within the chip differ in the maximum frequency that they can support. In the future, this effect is expected to grow. For example, at 32nm, it is estimated that the difference in frequency between the cores in a 20-core chip may reach over 20% [32].

If we assume a multicore with a single frequency domain, the maximum frequency supported by the slowest core determines the frequency of the whole chip. In this case, Figure 4(a) shows the aging curves for the slowest core in the chip (P_1) and the fastest one (P_2). At *t*=0, the critical paths of the cores have different delays (C_0^1 and C_0^2). As the chip is exercised, processors age. At the end of the expected service life, P_1's critical path delay reaches the clock period used by the multicore chip, namely τ_0.

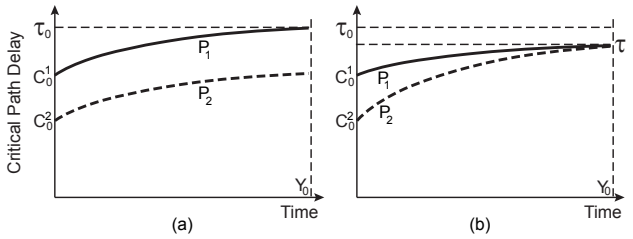


Figure 4: Hiding the effects of aging in a multicore with aging-driven application scheduling.

To hide the effects of aging, we propose to schedule applications on the different cores of the chip in an *aging-driven* manner. Specifically, we like the schedule to be such that the slowest core ages the slowest, while the faster cores age faster. The net result is shown in Figure 4(b). Since the slowest core has aged less than before, and it alone determines the chip's frequency, we are able to clock the chip with a period $\tau < \tau_0$. The faster cores have aged more but their aging is *hidden*.

To accomplish this behavior, recall from Table 1 that cores age faster if their *T* is high and if their logic switches frequently (they have a high α). This suggests scheduling hot, high-activity applications on the fast cores. Specifically, to simplify the implementation, we propose to measure the average *T* of applications as they run and, to the extent possible, steer high-*T* applications to the fast cores and low-*T* ones to the slow cores.

4.2.2 Implementation Issues

To implement this technique, we approximately measure the average *T* of an application on the fly, using per-core *T* sensors. Also, we need the ranking of cores according to the frequency they support. This can be obtained by running a simple test at regular periods, e.g., every month. The test is a suite of programs together with a check for the correct results, that is run on each core at increasing frequencies. As cores detect errors, the system constructs a ranking of cores according to their frequency. From then on until the next test, when multiple applications need to be scheduled, they are rank-ordered according to their *T* and assigned to processors that are rank-ordered according to their speed — giving higher-*T* applications to faster processors.

Note that the faster processors are likely to be the leakier ones and, as a result, this scheduling algorithm increases leakage. While this effect reduces the benefits obtained, we still expect the overall gains to be significant.

To estimate how much guardband the chip consumes with this technique, we proceed as follows. The baseline fraction of delay increase due to aging (S_0 in Figure 3(c)) assumes a certain average *T* of use — say T_{avg}. Under aging-driven scheduling, we size the fraction of delay increase (S_d in Figure 3(c)) assuming that the slowest core will be used at a lower average *T* — say T_{avg_slow}. With this S_d estimate, we can now compute the Simplification factor C_d/C_0 that we can rely on (if we desire a less refined design) or estimate the higher frequency f_f that we can use (if we desire a higher frequency).

The manufacturer can have a different T_{avg_slow} estimate for each different type of processor usage, and size guardbands appropriately. For example, for desktop multicores, where the different cores in the chip typically run different tasks and several cores are often idle, T_{avg_slow} is significantly lower than T_{avg}. In multicores for the server or supercomputing market, the cores in the chip are likely to run more similar tasks and be busy more often. In this case, T_{avg_slow} is closer to T_{avg}. In all cases, the periodic tests of processor speeds described above, or aging sensors such as those in [3, 5, 6, 28] can indicate if guardbands are being consumed at an usually high rate. If so, the system can trigger the techniques to slow down aging described in Section 4.3 or reduce the processor frequency.

Finally, although we focus on a multicore with a single frequency domain, aging-driven scheduling also works for multicores with multiple frequency domains. The same idea can be used, namely concentrate the aging mostly on the cores that, due to variation, are endowed with longer guardbands — i.e., the faster cores. However, in such an environment, there is a richer set of tradeoffs to be made. An analysis of such an environment is beyond our scope.

4.3 Slowing Down Aging: Chip-Wide ASV/ABB

4.3.1 Description of the Techniques

According to Table 1, two important parameters that determine the aging rate are $(V_{dd} - Vt)$ and *T*. To slow down aging, we propose to apply in a *chip-wide* manner, one of two techniques that directly affect these parameters, namely Adaptive Supply Voltage (ASV) and Adaptive Body Bias (ABB) [11].

In ASV, the chip's V_{dd} is slightly increased over its nominal value (a case we call ASV+) or slightly decreased (a case we call ASV-). Under ASV+, the gates become faster and spend more dynamic and static power. Under ASV-, the opposite occurs. This technique can reuse support for Dynamic Voltage and Frequency Scaling (DVFS), although the frequency is unchanged.

In ABB, a voltage is applied between the chip's substrate and the source (or drain) of the transistors. Depending on the voltage polarity, it either decreases the transistors' threshold voltage V_t (Forward Body Bias or FBB) or increases it (Reverse Body Bias or RBB). Under FBB, the gates become faster and consume more leakage power. Under RBB, the opposite occurs. ABB requires adding on-chip signal lines for the bias voltage. ABB is used in chips such as Intel's Xscale [12] and 80-core network-on-

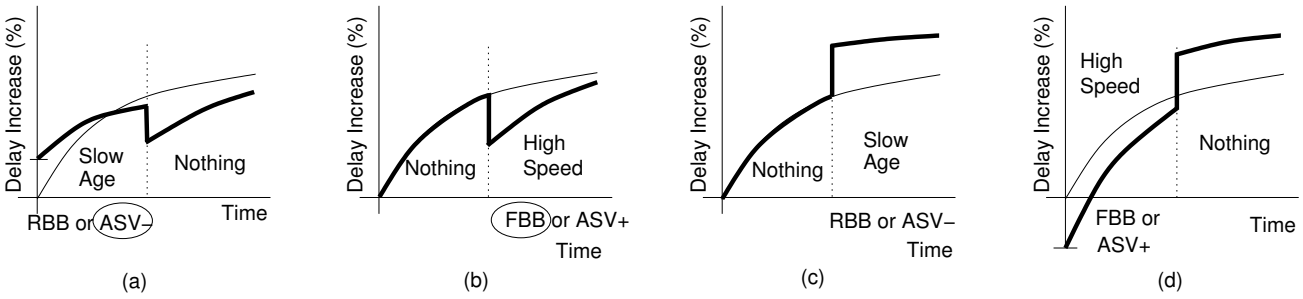


Figure 5: Applying techniques that change the aging rate during only part of the expected service life.

chip [35].

Consequently, we have two groups of techniques. The first one, which includes RBB and ASV-, increases the delay of critical paths but reduces the aging rate. The aging rate is reduced due to the decrease in V_{dd} (in ASV-), increase in V_t (in RBB), and decrease in *T* (in both, since both techniques save power). We call these techniques *SlowAge*.

The second group, which includes FBB and ASV+, reduces the delay of critical paths but increases the aging rate. The aging rate goes up due to the increase in V_{dd} (in ASV+), decrease in V_t (in FBB), and increase in *T* (in both, since both techniques increase power). We call these techniques *HighSpeed*.

Figure 6(a) shows the impact of the *SlowAge* techniques on the critical path delay of a core. The original critical path delay curve is shown in dashes. At *t*=0, *SlowAge* increases the core's critical path delay by a *Slowdown* amount over C_0. However, it also reduces the aging rate and, therefore, as the core is exercised, the critical path delay increases with a *lower slope* than before. Depending on the *SlowAge* technique's parameters, the critical path delay at the end of the service life (t=Y_0) may be lower than before (*Favorable Case* in the figure) or higher (*Unfavorable Case*).

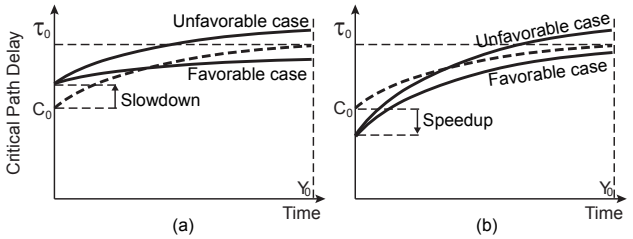


Figure 6: Changing the critical path delay in a core with *SlowAge* (a) and *HighSpeed* (b).

Figure 6(b) shows the impact of the *HighSpeed* techniques. At *t*=0, *HighSpeed* reduces the core's critical path delay by a *Speedup* amount. However, it also increases the aging rate and, therefore, as the core is used, the critical path delay increases with a *higher slope* than before. At t=Y_0, the core's critical path delay may be lower than before (*Favorable Case*) or higher (*Unfavorable Case*).

Since both *SlowAge* and *HighSpeed* techniques can potentially slow down aging, we now examine when to apply them and the difference between the effectiveness of ASV and ABB.

4.3.2 When To Apply These Techniques

A key observation is that these techniques impact (i) the aging rate and (ii) the critical path delay *differently*. Specifically, they impact the aging rate strongly at the beginning of the processor lifetime, and little toward the end. In contrast, they impact the delay more uniformly across time.

As an example, consider ASV+. From Equation 2, it can be shown that increasing V_{dd} increases the aging rate (i.e., the slope of the curve in Figure 3(a)) by roughly the same fraction irrespective of when it is applied. However, as shown in the figure, the slope changes with time — it is highest at *t*=0 and decreases with time. Consequently, the absolute impact of ASV+ on the aging rate is higher at the beginning than toward the end of the service life. On the other hand, Equation 1 shows that increasing V_{dd} reduces the critical path delay by the same amount irrespective of when it is applied — modulo the fact that V_t changes with time.

A consequence of this is that it is best to apply *SlowAge* techniques toward the beginning of the service life. At that time, they reduce the aging rate the most and, therefore, slow down aging the most. Toward the end of the service life, when they do not matter anyway, they can be disabled, and their contribution to lengthening the critical path delay mostly disappears. On the other hand, it is best to apply *HighSpeed* techniques only toward the end of the service life. At that time, they still reduce the critical path delay, while they increase the aging rate the least.

Figure 5 qualitatively shows the effect of applying these techniques during only part of the service life. We show their impact on the delay increase over C_0. In Figures 5(a) and (b), they are applied at their best time. Specifically, in Figure 5(a), *SlowAge* is applied in the beginning of the service life and no technique toward the end. When *SlowAge* is applied, it slows aging so much that, when it is removed, the critical paths are much faster than in the baseline case. After that, the remaining aging does not increase the delay to baseline values.

In Figure 5(b), we apply no technique at the beginning and *HighSpeed* toward the end. By the time we apply *HighSpeed*, the aging rate has substantially saturated. Consequently, *HighSpeed* increases the aging rate only slightly, while it still reduces the critical path delay.

In Figures 5(c) and (d), these techniques are applied at their worst time. Specifically, in Figure 5(c), *SlowAge* is applied toward the end — when the aging rate is small. In this case, the reduced aging rate cannot make up for the increase in critical path delay. In Figure 5(d), *HighSpeed* is applied at the beginning. By the time *HighSpeed* is removed, the critical paths have aged substantially, and they are slower than they would be in baseline conditions.

Finally, the best case involves applying a *SlowAge* technique in the beginning of the service life and a *HighSpeed* one toward the end.

4.3.3 ASV Versus ABB

While either ASV or ABB can be used to implement *SlowAge* and *HighSpeed*, they have different properties. Specifically, for a fixed change in critical path delay, ASV changes the aging rate

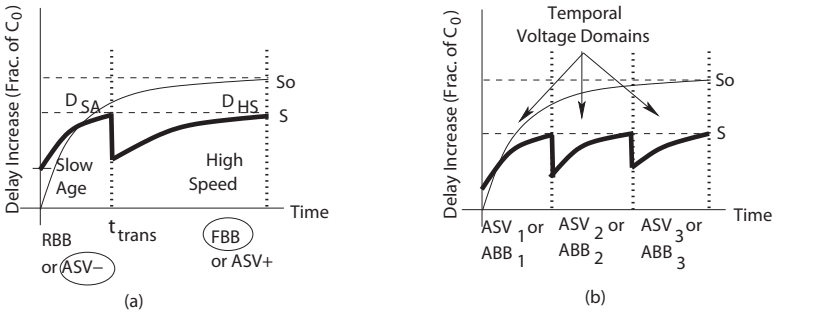


Figure 7: Applying different techniques in different epochs.

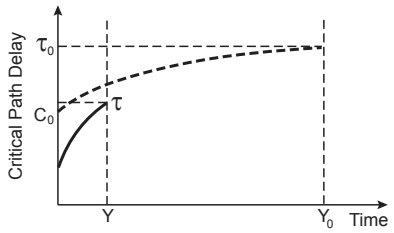


Figure 8: Configuring a core for a shorter service life.

more than ABB. Indeed, from Equation 1, it can be shown that, for the same change in aging rate, V_t changes due to ABB have a greater impact on delay than V_{dd} changes due to ASV. In addition, the overall impact of ASV on total power is higher than that of ABB — directly affecting *T* and, therefore, the aging. Overall, therefore, for a fixed change in delay, the aging rate is affected more with ASV.

Consequently, when applying a *SlowAge* technique as in Figure 5(a), we prefer ASV- over RBB — since ASV- reduces the aging rate more. On the other hand, when applying a *HighSpeed* technique as in Figure 5(b), we prefer FBB over ASV+ — since FBB reduces the critical path delay more. The best techniques are circled in Figures 5(a)-(b).

4.3.4 Implementation: Non-Linear Optimization

We propose to apply a *SlowAge* technique in an epoch at the beginning of the service life and a *HighSpeed* technique in an epoch toward the end. We need to find an optimal Transition Time (t_{trans}) where the *SlowAge* technique is disabled and the *HighSpeed* one is enabled (Figure 7(a)). Such optimal t_{trans} is one for which the maximum critical path delay increase (*S* in Figure 7(a)) is minimized.

There may be multiple optimal t_{trans}, and they are not necessarily at half the service life of the processor. At these points, the critical path delay increase under *SlowAge* (D_{SA}) must be equal to the delay increase under *HighSpeed* at the end of the service life (D_{HS}). This is shown in Figure 7(a). To see why, consider a counter-example where the delay increase at the end of the *SlowAge* epoch (D_{SA}) is higher than D_{HS}. In this case, we can slightly reduce the duration of *SlowAge* application, which will reduce D_{SA}. At the same time, D_{HS} will go up slightly, because more aging now takes place in the *HighSpeed* epoch. Overall, the result will be a lower maximum delay increase *S* for the processor. A symmetric argument holds for the case when D_{SA} is lower than D_{HS}.

To find one set of optimal values for t_{trans}, V_{dd_SA}, and V_{dd_HS} (or, under ABB, V_{bb_SA} and V_{bb_HS}), we proceed as follows. We generate the aging function $\Delta V_t()$ for each of the two epochs from the equations in Sections 2.1 and 2.2. Then, we plug $\Delta V_t()$ into the alpha power law (Equation 1) and, from there, build the delay of the critical paths. Finally, we constrain the delays of the critical paths at the end of the epochs to be $D_{SA} = D_{HS}$, and we minimize D_{SA} using the IPOpt [37] non-linear optimization package. The result of the algorithm is t_{trans}, the pair of V_{dd} (or V_{bb}), and the guardband increase needed *S*.

We envision this computation to be performed by the chip manufacturer. The *SlowAge* techniques are enabled from the beginning until t_{trans}. After that, the *HighSpeed* techniques are enabled until the end of the service life. As indicated before, it is possible that the working conditions of the chip differ markedly from those expected by the manufacturer. In this case, aging sensors in the chip such as those in [3, 5, 6, 28] detect if guardbands are about to be consumed too early. When this happens under *SlowAge*, the system triggers an early transition to *HighSpeed*; when it happens under *HighSpeed*, the system reduces the frequency.

To slow down aging even more, the algorithm presented can be easily extended to have more than two epochs. The idea is shown in Figure 7(b). Each epoch has a different V_{dd} (or V_{bb}). As the processor ages, the environment changes from strong *SlowAge* to weak *SlowAge*, to weak *HighSpeed*, and to strong *HighSpeed*. Effectively, we are having chip-wide *temporal* voltage domains — without changing the frequency. The result is a lower guardband increase needed *S*.

4.4 Consolidating Aging: Configuring Cores for a Shorter Service Life

Often, a user ends up discarding the processor chip before the expiration of the expected service life. In this case, significant available performance may be wasted. If we know when the chip will be discarded, we can use Facelift to "shift" some performance from the unused portion of the chip's lifetime to the used time.

This case is shown in Figure 8. The dashed curve shows the baseline critical path delay. We want to discard the chip at *t=Y*. Our goal is to apply *SlowAge* and *HighSpeed* techniques so that, during $0 \leq t \leq Y$, the critical path delay remains below a value τ that is as low as possible — allowing us to cycle the processor at the very high frequency $f=1/\tau$ all along.

The approach we use is similar to that of Section 4.3.4. We record the short service life desired (*t=Y*), and run the non-linear optimization technique to find the parameters of two (or more) *SlowAge* and *HighSpeed* epochs, the transition times, and the maximum guardband needed ($\tau - C_0$). The resulting critical path delay evolution is shown in Figure 8 as a solid curve.

5 Evaluation Setup

We evaluate our techniques for hiding, slowing down, and consolidating aging using the SESC [23] execution-driven simulator. In the following, we describe the models that we use.

5.1 Architecture Modeled

We model a chip multiprocessor at 32nm with 16 cores running at a baseline frequency of 4 GHz. The cores are out-of-order, 4-issue, and similar to the Alpha 21264. Each core has private L1 and L2 caches. The chip has a single frequency and voltage domain. The architecture parameters are shown in Ta-

Architecture	Aging and Variation
Chip: 16-core multicore, 32nm, 1V	Expected processor service life: 7 years
Frequency: 4GHz, single frequency and voltage domain	*Low Wearout* aging rate: 10% in 7 years
Cores: out-of-order, 4-issue, like Alpha 21264	*High Wearout* aging rate: 25% in 7 years
On-chip network: 4x4 2-D torus, 6 cyc hop latency	V_{t0}: 200mV at 80 ^{o}C
Private D-L1, I-L1: 64KB, 2-way, 64B line, 2 cyc roundtrip	Max ΔV_{dd} for ASV: $\pm$0.1V; Max ΔV_t for ABB: $\pm$75mV
Private L2: 2MB, 8-way, 64B line, 8 cyc roundtrip	V_t var: σ/μ=0.12, ϕ=0.3; L_{eff} var: σ/μ=0.06, ϕ=0.3
Memory: 400 cycle roundtrip	Number of chips/experiment: 100
T_{max}: 120 ^{o}C	Average core *T* running apps: 47–92 ^{o}C

Table 2: Architecture modeled. All latencies are given in processor cycles.

ble 2. Our simulator is enhanced with dynamic power models based on Wattch [9] and CACTI [33]. We also model static power with models based on HotLeakage [40] and temperature with HotSpot [27]. We control that the temperature never exceeds 120 ^{o}C.

5.2 Modeling Critical Paths

We do not have access to detailed information on the structure and distribution of a real processor's critical paths. For this reason, we design a simple model for our experiments. The model was outlined in Section 3.2. In pipeline stages with logic structures (e.g., the execution unit), we model critical paths as chains of FO4 inverters. In stages with memory structures (e.g., the cache access), we model critical paths as stretching from the decoder of the structure, to the wordline, pass transistor, bitline, and sense amplifier. We use CACTI to determine the optimal sub-array sizes, physical layout, and cycle count of the structures.

We model each pipeline stage as having many, spatially-distributed critical paths. Specifically, we use Bowman *et al.*'s [8] estimate that a high-performance processor chip at our technology may have $\approx$ 50,000 critical paths. We distribute these paths uniformly on the area taken by the cores and L1 caches — we assume that the L2s and the interconnect do not have critical paths. Each pipeline stage gets critical paths of its type.

5.3 Modeling Aging

To our knowledge, there is no publicly-available validated information on expected service lifes and aging rates of processors. Consequently, we assume a range of values. Specifically, we assume that processor chips are designed for a 7-year service life [3], and evaluate two different aging rates, called *Low Wearout* and *High Wearout*. They increase the delay of the critical paths by 10% and by 25%, respectively, in 7 years — namely, G_0 in Figure 3(a) is 0.10 $\times C_0$ and 0.25 $\times C_0$, respectively. These numbers are similar to those assumed in the literature.

There are three other important parameters related to aging which, due to the lack of experimental data, are explored with ranges of values. The first one is the average fraction of time when PMOS transistors are in stress mode. This parameter determines the fraction of the time that NBTI aging is active. We examine a range between 10% and 90%, and a default value of 50%. The second parameter is NBTI's impact on transistor delay relative to HCI's impact — given two transistors of the correct types. We use a range between 1 and 10, and a default value of 3, as asserted by Bernstein *et al.*'s [4]. The final parameter is the average ratio of PMOS to NMOS transistors in critical paths. We use a range between 0.1 and 10, and a default value of 1. The last two parameters determine the relative impact of NBTI and HCI on aging. With the default parameters, we calibrate the A_{NBTI} and A_{HCI} constants from Sections 2.1 and 2.2 to induce Low and High Wearout as defined.

In *SlowAge* and *HighSpeed* techniques, when we apply ASV, we change V_{dd} by at most $\pm$0.1V; when we apply ABB, we apply voltages that change V_t by at most $\pm$75mV. These figures are shown in Table 2.

5.4 Modeling Process Variation

To model process variation, we use the VARIUS framework [25]. VARIUS models the within-die systematic and random variation of V_t and L_{eff}, which it characterizes with two statistical measures, namely σ/μ and ϕ. The latter measures the spatial correlation of the systematic component of variation. For these measures, we use values similar to those recommended by the authors, as shown in Table 2.

Using this model, we create chip-wide variation maps for V_t and L_{eff}. As we superimpose these maps and the temperature profiles obtained from the simulator on the multicore layout, we use Equation 1 to determine how variation impacts the delay of each gate of each critical path. The slowest of the critical paths in a processor determines the processor frequency. Finally, since variation is a statistical process, using the same σ/μ and ϕ parameter values, we create 100 variation profiles — which correspond to 100 different chips. Every experiment in Section 6 is performed on all of the 100 chips, and the average is reported. We found that including more than 100 chips in our experiments did not change the numbers appreciably.

5.5 Workloads Used

The application set consists of the 26 SPEC2000 int and fp applications. Each application is measured for 1B instructions, after skipping its initialization. For our experiments, we place the applications in random order in a circular queue. Then, we construct many 16-application workloads by consecutively selecting the next 16 applications in order from the queue. We assume that the load generated by each of these 16-application workloads prevails for 10 days. Consequently, to model the load in a 7-year service life, we simulate 3 (workloads/month) x 12 (months/year) x 7 (years) = 252 different workloads, and report the cumulative results. In the baseline multicore, the programs of each of these 16-application workloads are scheduled on the 16 cores randomly. We refer to this approach as *Random* scheduling.

Table 2 shows that the average temperature (*T*) of the cores running the applications ranges from 47 to 92 ^{o}C. This number is obtained with random scheduling. For each given core-application pair, we compute the average *T* across time and across all the modules in the processor.

6 Results

6.1 Hiding Aging with Aging-Driven Scheduling

To gain intuition, we take one sample chip and measure the increase in the delay of the chip-critical path due to aging under different application scheduling algorithms. We consider the base-

line Random scheduling and the proposed Aging-Driven scheduling (which we call *Sched*), both under High Wearout (Figure 9). We see that, with Random scheduling, the delay of the chip-critical path in this particular chip increases by ≈23% in 7 years. With *Sched*, the delay increase is only ≈14%, thanks to the aging-hiding effect of *Sched*.

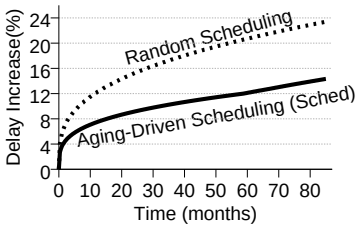


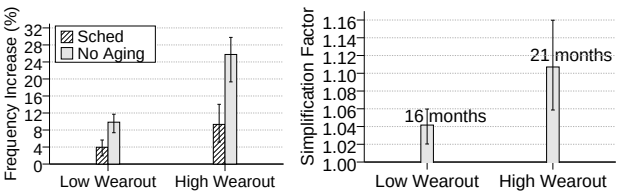
Figure 9: Increase in the delay of the chip-critical path under different scheduling algorithms.

Sched steers low-*T* applications to the slowest processor, which ages the least. Table 3 shows the *T* for the average processor-application pair under Random scheduling (top row) and for the slowest processor under *Sched* (bottom row). For a given pair, we measure the maximum *T* at any point in the execution and at any module in the processor (*Max(T)*) and the average *T* across the execution and modules (*Avg(T)*). From the table, we see a large difference between *T* under Random scheduling and in the slowest processor under *Sched*. This is the source of *Sched*'s ability to limit aging. It helps, of course, that this application mix has variance, as can be seen from the *T* ranges.

Environment	*Max(T)* (°C)		*Avg(T)* (°C)	
	Avg	Range	Avg	Range
Random Scheduling	92	47–116	77	47–92
Sched (Slowest Proc)	52	47–59	50	47–55

Table 3: Measured *T* in processor-application pairs.

One way to leverage *Sched* is to increase the multicore frequency and still retain the same service life. For this scenario, Figure 10(a) shows the frequency increase over the baseline enabled under Low and High Wearout. For comparison, each wearout level also includes a second bar with the frequency increase possible had there been no aging. Each bar shows the average of all 100 chips and the spread. The figure shows that *Sched* increases the frequency over Random by, on average, 4% and 9% under Low and High Wearout, respectively. Moreover, *Sched* regains 35–40% of the frequency loss due to aging (*No Aging* bars).



(a) (b)

Figure 10: Frequency increases (a) and Simplification factors (b) enabled by *Sched*.

The second way to leverage *Sched* is to spend less time refining the multicore design. For this scenario, Figure 10(b) shows the Simplification factors relative to the baseline under Low and High Wearout. As shown by the Simplification factors, the critical path delay of the slowest core in the chip can be extended at *t*=0 by 4% and 11% under Low and High Wearout, respectively — and still have enough guardband to complete the full service life. The chips can be designed as if they were expected to last for only a reduced service life. Such reduced service life is shown on top of the bars: it is only 16 and 21 months for Low and High Wearout, respectively. Compared to the 7-year baseline service life, this shows the remarkable aging-hiding impact of *Sched*.

6.2 Slowing Aging with Chip-Wide ASV or ABB

We take a sample chip and apply the non-linear optimization algorithm of Section 4.3.4 on the slowest processor of the chip to identify optimal *SlowAge* and *HighSpeed* epochs. Here, we only use ASV- and ASV+. Figure 11 shows the resulting effect on the delay of the chip-critical path under High Wearout. We see that, in the *SlowAge* epoch, the critical path delay starts-off longer, but it grows more slowly than in the baseline. At the 16-month point, the processor enters the *HighSpeed* epoch. The critical path delay drops significantly, at the cost of a slight increase in the slope of the critical path delay growth. However, since the aging rate is already small by now, the critical path delay at the end of the service life is much smaller than in the baseline.

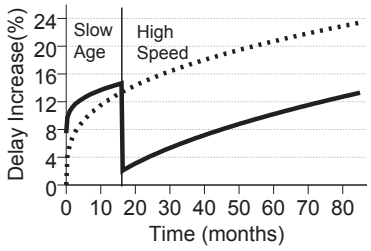
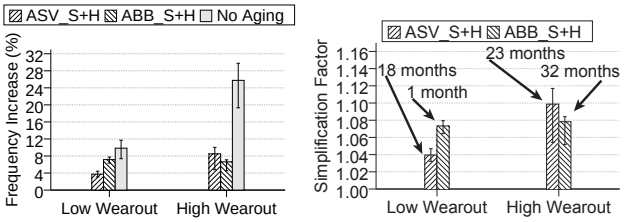


Figure 11: Increase in the delay of the chip-critical path with *SlowAge* and *HighSpeed* epochs.

We now apply aging-slowing techniques to the whole 100-chip batch to increase frequency or enable the use of a less refined design. Figure 12(a) shows the frequency increases enabled by ASV_{S+H} or ABB_{S+H} application under Low and High Wearout — where *S+H* stands for *SlowAge+HighSpeed*. The bars show the average and spread of the measurements. From the figure, we see that the frequency increases attained after slowing down aging are significant. Using ASV_{S+H}, we increase the average frequency by 4% and 9% for Low and High Wearout, respectively. Using ABB_{S+H}, the increase is 7% under both Low and High Wearout.



(a) (b)

Figure 12: Frequency increases (a) and Simplification factors (b) enabled by ASV_{S+H} or ABB_{S+H}.

ASV_{S+H} performs better under High Wearout and ABB_{S+H} under Low Wearout. This is because, as per Section 4.3.3, ASV is more effective at changing the aging rate, whereas ABB is more

effective at changing path delays. Consequently, when the aging rate is high, ASV_{S+H} performs better, while when the aging rate is low, ABB_{S+H} performs better.

If, instead, we choose to use a less refined design, Figure 12(b) shows the average Simplification factors and the equivalent target service lifes for the techniques. As shown in the figure, with these techniques, the chips can be designed as if they were expected to be used for only 1–32 months, on average — rather than the baseline 7 years.

6.3 Hiding and Slowing Down Aging

We now combine *Sched* with ASV_{S+H} or ABB_{S+H}. Note that the non-linear optimization algorithm works in the same way. Figure 13(a) shows the frequency increases enabled by *Sched*+ASV_{S+H} and *Sched*+ABB_{S+H} under Low and High Wearout. For comparison, we also show *Sched* and *No Aging*. We see that *Sched*+ASV_{S+H} enables an increase of the frequency by, on average, 8% and 14% under Low and High Wearout, respectively. *Sched*+ABB_{S+H} is slightly better with 11% and 15% increases. Comparing these bars to *No Aging*, we see that the combination of aging-hiding and slowing techniques recovers 54–110% of the frequency lost to aging. Note that, in one case, these techniques do even better than *No Aging*. This is because, thanks to the power spent by ABB, the critical paths are sped-up more than they are slowed down by aging. Finally, the bars show that aging-hiding and aging-slowing techniques combine well.

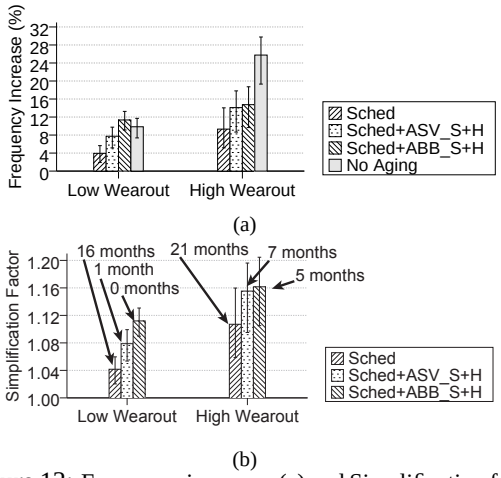


Figure 13: Frequency increases (a) and Simplification factors (b) enabled by the combination of aging-hiding and aging-slowing techniques.

Figure 13(b) shows the Simplification factors. From the *Sched*+ASV_{S+H} and *Sched*+ABB_{S+H} bars, we see that the critical path delay of the slowest core in the chip can be extended at t=0 by 8–11% under Low Wearout and 15–16% under High Wearout. This enables a chip designed for 1 month under Low Wearout or 5–7 months under High Wearout to last for 7 years.

6.4 Consolidating for a Shorter Service Life

We now configure the chips for 1- and 3-year expected service lifes. Figure 14 shows the frequency increases enabled by ASV_{S+H}, *Sched*+ASV_{S+H}, ABB_{S+H}, and *Sched*+ABB_{S+H}. For comparison, we also show the frequency increases for 7 years. We find that *Sched*+ABB_{S+H} is the best performing technique for both wearout environments. The key reason behind this is that ABB is a good *HighSpeed* technique, better able than ASV to speed up circuits with a lesser cost in aging rate.

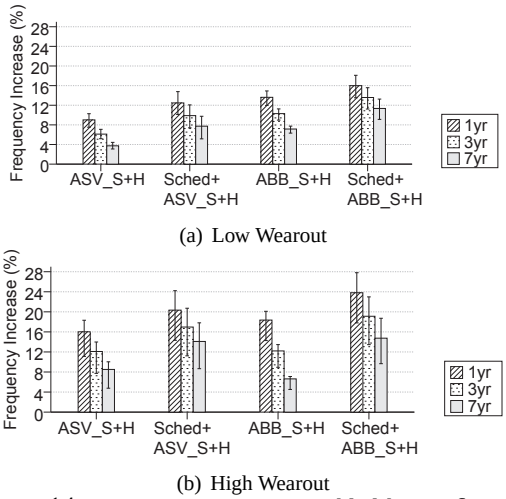


Figure 14: Frequency increases enabled by configuring the chips for shorter expected service lifes.

For 1- and 3-year expected service lifes, these techniques enable sizable frequency increases. For example, *Sched*+ABB_{S+H} enables frequency increases of, on average, 14–16% for Low Wearout and 19–24% for High Wearout. To realize such substantial frequency increases, the chip may require an upgrade in its power dissipation capability — a topic beyond this paper's scope. Overall, these results show that, if we are willing to discard chips early, we can extract much more performance during their short service life.

6.5 Sensitivity Analysis

To ensure that our techniques are applicable for a wide range of conditions, we briefly describe four sensitivity experiments we performed. Figure 15 shows the increase in the critical path delay of the slowest processor in the chip if the fraction of time that its PMOS transistors are under stress changes. We had calibrated the Low Wearout environment so that if such a fraction of time is 50%, the processor slows down 10% after 7 years. We see that if the fraction is 10–90%, the slowdown changes to 2–20%.

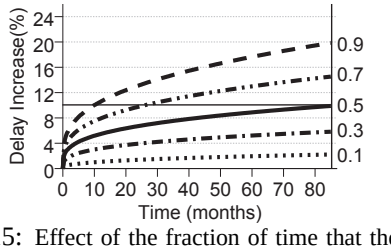
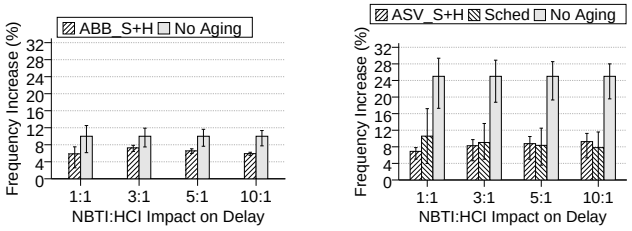


Figure 15: Effect of the fraction of time that the PMOS transistors are under stress under Low Wearout.

Figure 16 shows the frequency increases enabled as we vary NBTI's impact on transistor delay relative to HCI's impact—given transistors of the correct types. We examine the range 1:1 to 10:1, which includes the default value of 3:1. The total delay increase after 7 years is fixed at 10% and 25% for Low and High Wearout. We only show two of the most effective techniques: ABB_{S+H} under Low Wearout, and *Sched* and ASV_{S+H} under High Wearout. Overall, this parameter has little impact.

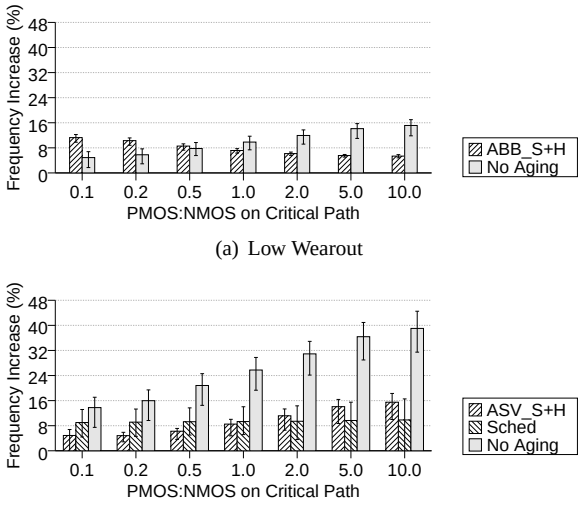
Figure 17 shows the frequency increases enabled as we vary the ratio of PMOS to NMOS transistors in the critical path. We



(a) Low Wearout (b) High Wearout

Figure 16: Frequency increases enabled for different NBTI:HCI impact on delay for a constant total aging.

examine the range 0.1 to 10, which includes the default value of 1. Since NBTI's impact on delay is higher than HCI's, as the ratio increases, the total aging increases. Hence, the *No Aging* bars go up. The ABB_{S+H} bars go down because ABB_{S+H} is less effective with higher aging rates. The ASV_{S+H} bars go up because the opposite is true for ASV_{S+H}. *Sched* stays constant because it saves the same amount of aging for PMOS and NMOS.



(a) Low Wearout

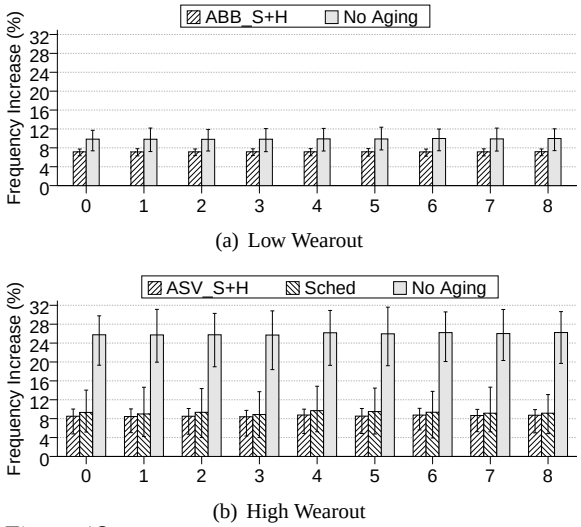
(b) High Wearout

Figure 17: Frequency increases enabled for different PMOS:NMOS transistor ratios in the critical path.

Figure 18 shows the effect of changing the order in which the applications are placed in the circular queue for the experiments (Section 5.5). This results in different workload compositions for our experiments. The figure compares the baseline order (bars *0*) to 8 other permutations. The differences are negligible.

7 Related Work

Srinivasan *et al.* [29, 30] examine the related problem of lifetime reliability of processors. Their work is different than ours in terms of the problem looked at, the goals, and the approach taken. They focus on the problem of Mean Time To Failure (MTTF) of processors due to aging mechanisms. The value of this MTTF is typically multiple times the expected service life of a processor, since one should expect that only a negligible fraction of processors will fail during their service life. Their goal is to estimate at run time the MTTF based on operating conditions since the processor was manufactured, and then extend the MTTF or attain it more cost-effectively. Their approach is to monitor application behavior and, based on that, dynamically adapt the processor (*V*, *f*,



(a) Low Wearout

(b) High Wearout

Figure 18: Frequency increases enabled for different application permutations.

or microarchitecture structures) to maintain its reliability target in MTTF — an approach called Dynamic Reliability Management.

Rather than focusing on failures, we instead look at the problem of progressive processor slowdown with time due to aging during the expected service life of the processor. Our goal is to reduce the rate of guardband consumption, so that processors can be designed more cost-effectively — namely, they need less design tuning to work or, for the same tuning, can be clocked at a higher, fixed frequency. Finally, our approach is to hide the effects of aging with aging-driven job scheduling, and slow down aging with a one-time change in V_{dd} or V_t (or a few changes, always in the same direction).

Blome *et al.* [5, 6], Agarwal *et al.* [3], and Smolens *et al.* [28] propose circuits to detect when critical paths have slowed down due to aging. These are "aging sensors" that can be used to initiate the replacement of components. Facelift can use these sensors to improve its effectiveness.

Ramakrishnan *et al.* [22] reduce the NBTI-induced wearout in FPGAs by loading NBTI-reversing bit patterns into devices (i.e., a logic 1 in the gates of PMOS transistors) during idle periods. Such patterns reverse the aging as described in Section 2.1. Abella *et al.* [1] use the same technique in a processor. They discuss approaches to insert the desired bit patterns into several microarchitectural structures. Most recently, Shin *et al.* [26] propose a similar approach for SRAM caches. It proactively puts PMOS transistors from cache arrays in recovery mode. The scheme requires moving data to a spare cache array. While all these techniques are effective and transparent to software layers, if they are applied widely across processor structures, they are likely to be intrusive to the processor design and may have performance implications.

Facelift does not try to change the stress time of PMOS transistors. Instead, it changes high-level parameters such as application scheduling, V_{dd}, and V_t, that can be easily manipulated architecturally. Our work is orthogonal to theirs, and both can be used simultaneously.

Finally, Abella *et al.* [2] propose an improved design of memory cells that is resilient to NBTI. In their design, the stress time of PMOS transistors is reduced by construction.

8 Conclusions

This paper presented *Facelift*, a framework to address the gradual slowdown that processors experience during their service life. The paper made two contributions. First and foremost, it presented a set of techniques to (i) hide the effects of aging in a multicore, (ii) slow down aging, and (iii) gainfully configure the chip for a short service life. A second contribution was to show how the resulting shorter guardband needed can be used to either design a less refined processor version or to clock the processor at a higher frequency.

Facelift hides the effects of aging in a multicore by steering jobs in an aging-driven manner: high-temperature ones to fast cores and low-temperature ones to slow cores. Keeping the slow cores cooler enables the chip to appear to age less. Facelift slows down aging by making *chip-wide* changes to V_{dd} or V_t in two (or more) time epochs, carefully balancing the impact on the aging rate and on the critical path delays. We use a non-linear optimization algorithm to select the optimal voltages and the optimal epoch-transition point. Finally, Facelift configures a chip for a short service life by "shifting" performance from the unused lifetime portion to the used one.

Overall, our results showed that Facelift leads to more cost-effective multicore designs. For example, we can take a multicore designed for a 7-year service life and, by hiding and slowing down aging, enable it to run, on average, at a 14–15% higher frequency. Alternatively, we can design a multicore for a 5 to 7-month service life and use it for 7 years. Finally, the implementation of the Facelift techniques is very simple.

References

[1] J. Abella, X. Vera, and A. González. Penelope: The NBTI-aware processor. In *Int. Symp. on Microarchitecture*, December 2007.

[2] J. Abella, X. Vera, O. Unsal, and A. Gonzalez. NBTI-resilient memory cells with NAND gates for highly-ported structures. In *Workshop on Dependable and Secure Nanocomputing*, June 2007.

[3] M. Agarwal, B. Paul, and S. Mitra. Circuit failure prediction and its application to transistor aging. In *VLSI Test Symposium*, May 2007.

[4] K. Bernstein et al. High-performance CMOS variability in the 65-nm regime and beyond. In *IBM Journal of Res. and Dev.*, 2006.

[5] J. Blome, S. Feng, S. Gupta, and S. Mahlke. Online timing analysis for wearout detection. In *Workshop on Architectural Reliability*, December 2006.

[6] J. Blome, S. Feng, S. Gupta, and S. Mahlke. Self-calibrating online wearout detection. In *Inter. Symp. on Microarch.*, December 2007.

[7] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6), 2005.

[8] K. Bowman, S. Duvall, and J. Meindl. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE Journal of Solid-State Circuits*, 2002.

[9] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *International Symposium on Computer Architecture*, June 2000.

[10] G. Chen, M. F. Li, C. H. Ang, J. Z. Zheng, and D. L. Kwong. Dynamic NBTI of p-MOS transistors and its impact on MOSFET scaling. In *IEEE Electron Device Letters*, December 2002.

[11] T. Chen and S. Naffziger. Comparison of adaptive body bias (ABB) and adaptive supply voltage (ASV) for improving delay and leakage under the presence of process variation. *IEEE Transactions on VLSI Systems*, October 2003.

[12] L.T. Clark et al. An embedded 32-b microprocessor core for low-power and high-performance applications. *IEEE Journal of Solid-State Circuits*, 2001.

[13] E. Humenay, D. Tarjan, and K. Skadron. Impact of process variations on multicore performance symmetry. In *Conference on Design, Automation and Test in Europe*, April 2007.

[14] K. Kang et al. Estimation of statistical variation in temporal NBTI degradation and its impact in lifetime circuit performance. In *Int. Conf. on Computer-Aided Design*, October 2007.

[15] N. Kimizuka et al. The impact of bias temperature instability for direct-tunneling ultra-thin gate oxide on MOSFET scaling. In *Symposium on VLSI Technology Digest of Technical Papers*, 1999.

[16] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. An analytical model for negative bias temperature instability. In *International Conference on Computer-Aided Design*, November 2006.

[17] X. Liang and D. Brooks. Mitigating the impact of process variations on CPU register file and execution units. In *International Symposium on Microarchitecture*, December 2006.

[18] D. Marculescu and E. Talpes. Variability and energy awareness: A microarchitecture-level perspective. In *Design Automation Conference*, June 2005.

[19] J. W. McPherson. Reliability challenges for 45nm and beyond. In *Design Automation Conference*, July 2006.

[20] S. Ogawa and N. Shiono. Generalized diffusion-reaction model for the low-field charge build up instability at the $Si - SiO_2$ interface. In *Phys. Rev. B.*, February 1995.

[21] B. C. Paul et al. Negative bias temperature instability: Estimation and design for improved reliability of nanoscale circuits. In *Trans. on Computer-Aided Design of Integrated Circuits and Systems*, April 2007.

[22] K. Ramakrishnan, S. Suresh, N. Vijaykrishnan, M. J. Irwin, and V. Degalahal. Impact of NBTI on FPGAs. In *International Conference VLSI Design*, January 2007.

[23] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, K. Strauss, S. R. Sarangi, P. Sack, and P. Montesinos. SESC Simulator, January 2005. http://sesc.sourceforge.net.

[24] T. Sakurai and R. Newton. Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas. *IEEE Journal of Solid-State Circuits*, 1990.

[25] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. VARIUS: A model of process variation and resulting timing errors for microarchitects. In *IEEE Transactions on Semiconductor Manufacturing*, February 2008.

[26] J. Shin, V. Zyuban, P. Bose, and T. M. Pinkston. A proactive wearout recovery approach for exploiting microarchitectural redundancy to extend cache SRAM lifetime. In *International Symposium on Computer Architecture*, June 2008.

[27] K. Skadron et al. Temperature-aware microarchitecture. In *International Symposium on Computer Architecture*, June 2003.

[28] J. Smolens, B. T. Gold, J. C. Hoe, B. Falsafi, and K. Mai. Detecting emerging wearout faults. In *Workshop on Silicon Errors in Logic – System Effects*, April 2007.

[29] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The case for lifetime reliability-aware microprocessors. In *International Symposium on Computer Architecture*, June 2004.

[30] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. Exploiting structural duplication for lifetime reliability enhancement. In *International Symposium on Computer Architecture*, June 2005.

[31] E. Takeda, C. Y. Yang, and A. Miura-Hamada. *Hot-Carrier Effects in MOS Devices*. Academic Press, 1995.

[32] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *International Symposium on Computer Architecture*, June 2008.

[33] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. Jouppi. CACTI 5.1. Technical Report HPL-2008-20, HP Labs, April 2008.

[34] Y. Uraoka et al. Hot carrier effect in ultrathin gate oxide metal oxide semiconductor field effect transistor. In *Japanese Journal of Applied Physics*, 2005.

[35] S. Vangal et al. An 80-tile 1.28 TFLOPS network-on-chip in 65nm CMOS. In *Int. Solid-State Circuits Conference*, Feb. 2007.

[36] R. Vattikonda, W. Wang, and Y. Cao. Modeling and minimization of PMOS NBTI effect for robust nanometer design. In *Design Automation Conference*, July 2006.

[37] A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. In *Mathematical Programming*, 2006.

[38] W. Wang et al. Compact modeling and simulation of circuit reliability for 65-nm CMOS technology. In *IEEE Transactions on Device and Materials Reliability*, December 2007.

[39] W.-K. Yeh et al. Temperature dependence of hot-carrier-induced degradation in 0.1 μm SOI nMOSFETs with thin oxide. In *IEEE Electron Device Letters*, July 2002.

[40] Y. Zhang et al. HotLeakage: A temperature-aware model of subthreshold and gate leakage for architects. Technical Report CS-2003-05, Univ. of Virginia, 2003.