Maestro: Orchestrating Lifetime Reliability in Chip Multiprocessors

Shuguang Feng Shantanu Gupta Amin Ansari Scott Mahlke

Advanced Computer Architecture Laboratory
University of Michigan
Ann Arbor, MI 48109
{shoe, shangupt, ansary, mahlke}@umich.edu

ABSTRACT

As CMOS feature sizes venture deep into the nanometer regime, wearout mechanisms including negative-bias temperature instability and time-dependent dielectric breakdown can severely reduce processor operating lifetimes and performance. This paper presents an introspective reliability management system, Maestro, to tackle reliability challenges in future chip multiprocessors (CMPs) head-on. Unlike traditional approaches, Maestro relies on low-level sensors to monitor the CMP as it ages (introspection). Leveraging this real-time assessment of CMP health, runtime heuristics identify wearout-centric job assignments (management). By exploiting the complementary effects of the natural heterogeneity (due to process variation and wearout) that exists in CMPs and the diversity found in system workloads, Maestro composes job schedules that intelligently control the aging process. Monte Carlo experiments show that Maestro significantly enhances lifetime reliability through intelligent wear-leveling, increasing the expected service life of a population of 16-core CMPs by as much as 38% compared to a naive, round-robin scheduler. Furthermore, in the presence of process variation, Maestro's wearout-centric scheduling outperformed both performance counter and temperature sensor based schedulers, achieving an order of magnitude more improvement in lifetime throughput – the amount of useful work done by a system prior to failure.

1. INTRODUCTION

In recent years, computer architects have accepted the fact that transistors become less reliable with each new technology generation [4]. As technology scaling leads to higher device counts, power densities and operating temperatures will continue to rise at an alarming pace. With an exponential dependence on temperature, faults due to failure mechanisms like negative-bias temperature instability (NBTI) and time-dependent dielectric breakdown (TDDB) will result in ever-shrinking device lifetimes. Furthermore, as process variation (random + systematic) and wearout gain more prominence in future technology nodes, fundamental design assumptions will become increasingly less accurate. For example, the characteristics of a core on one part of a chip multiprocessor (CMP) may, due to manufacturing defects, only loosely resemble an *identically designed* core on a different part of the CMP [26, 23]. Even the behavior of the same core can be expected to change over time as a result of age-dependent degradation [18, 25].

In light of this uncertain landscape, researchers have begun investigating dynamic thermal and reliability management (DTM and DRM). Such techniques hope to sustain current performance improvement trends deep into the nanometer regime, while maintaining the levels of reliability and life-expectancy that consumers have come to expect, by hiding a processor's inherent susceptibility to failures and hotspots. Some recent proposals rely on a combination of thread scheduling and dynamic voltage and frequency scaling (DVFS) to recover performance lost to process variation [23, 26]. Others implement intelligent thermal management policies that can extend processor lifetimes and alleviate hotspots by minimizing and bounding the overall thermal stress experienced by a core [16, 17, 9, 7]. There have also been efforts to design sophisticated circuits that tolerate faults and adaptive pipelines with flexible timing constraints [10, 24]. Although many DTM schemes actively manipulate job-to-core assignments to avoid thermal emergencies, most existing DRM approaches only *react* to faults, tolerating them as they develop.

In contrast, Maestro takes a proactive approach to reliability. To the first order, Maestro performs fine-grained, module-level wear-leveling for many-core CMPs. Although analogous to wear-leveling in flash devices, the challenge of achieving successful wear-leveling transparently in CMPs is considerably more difficult. Left unchecked, wearout causes all structures within a core to age and eventually fail. However, due to process variation, not all cores (or structures) will be created equal. Every core will invariably possess some microarchitectural structures that are more "damaged" (more susceptible to wearout) than others [24, 23]. Performing post-mortems on failed cores (in simulations) often reveals that a single microarchitectural module, which varies from core to core, breaks down long before the rest. Maestro extends the life of these "weak" structures, their corresponding cores, and ultimately the CMP by ensuring uniform aging with scheduling-driven wear-leveling across all levels of the hierarchy.

Maestro dynamically formulates wearout-centric schedules, where jobs are assigned to cores such that cores do not execute workloads that apply excessive stress to their weakest modules (i.e., a floating-point intensive thread is not bound to a core with a weakened floating-point adder). This accomplishes *local wear-leveling* at the core level, avoiding failures induced by a single weak structure. When two cores both have a strong affinity for the same job, a heuristic, which enforces *global wear-leveling* at the CMP level determines which core is given priority. Typically, unless there is a substantial negative impact on local wear-leveling, deference is given to the weaker of the two cores. This ensures that, when necessary, stronger cores are allowed to execute less desirable jobs in order to postpone failures in weaker cores (details in Section 3.2).

By leveraging the natural, module-level diversity in application thermal footprints (Section 2.1), Maestro has finer-grained control over the aging process than a standard core-level DVFS approach, without any of the attendant hardware/design overheads. Given the complex nature of wearout degradation, Maestro departs from the conventional reliance on static analysis to project optimized schedules. Instead, the condition of the underlying CMP hardware is continuously monitored, allowing Maestro to dynamically refine and adapt scheduling algorithms as the system ages. Architectures like those envisioned in [22], with low-level circuit sensors, can readily supply this real-time "health" monitoring.

Maestro offers two key benefits for future CMP systems. First, the fine-grained, local wear-leveling prevents unnecessary core failures, maximizing the life of *individual* cores. Longer lasting cores translates to more work that can be done over the life of the system. Second, it improves the ability of the system to sustain heavy workloads despite the effects of aging. Enforcing global wear-leveling maximizes the *number* of functional cores (throughout its useful life), which in turn maximizes the computational horsepower available to meet peak demands. With higher degrees of process variation on the horizon, premature core failures will make it increasingly more difficult to design and qualify future CMPs. However, by harnessing the potential of Maestro, proactive management will enable semiconductor manufacturers to provide chips with longer lifetimes as well as ensure that system performance targets are consistently met throughout that lifetime. The central contributions of this paper include:

- An evaluation of workload variability and its impact on reliability/wearout.
- An introspective system, Maestro, that utilizes low-level sensor feedback and application-driven wear-leveling to proactively manage lifetime reliability.
- The design and evaluation of two reliability-centric job scheduling algorithms.

2. SCHEDULING FOR DAMAGED CORES AND DYNAMIC WORKLOADS

Scheduling, in the context of this paper, refers to the process of assigning jobs to cores in a CMP, and is conceptually decoupled from the operating system (OS) scheduler. The schedulers proposed by microarchitects in the past typically resided in a virtualization layer (i.e., system firmware) that sits between the OS and the underlying hardware. At each scheduling interval, the OS supplies a set of jobs, J, to this virtualization layer, and it is the task of the low-level scheduler to bind the jobs to cores. Prior works have investigated techniques that leverage intelligent job scheduling to manage on-core temperatures or cope with process variation. However, none have studied the impact that wearout-centric scheduling alone can have on the evolution of aging within a core.

Embracing process variation and workload diversity, Maestro can enhance lifetime reliability without the extensive hardware support for adaptive body biasing (ABB) and adaptive supply voltage (ASV) required by other approaches [25]. The remainder of this paper targets TDDB and NBTI, which are expected to be the two leading causes of wearout-related failures in future technologies, but can be easily extended to address any progressive failure mechanisms that may emerge. Since both TDDB and NBTI are highly dependent on temperature, it is important to understand the thermal footprints of typical applications in order to appreciate the potential for reliability-centric scheduling. Section 2.1 examines the module-level thermal diversity seen across a set of SPEC2000 applications and Section 2.2 presents preliminary results quantifying the impact of this variation on processor lifetimes.

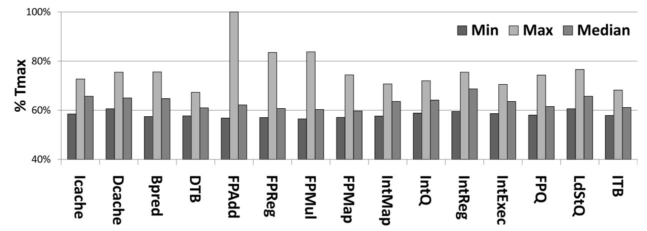


Figure 1: Variation of module temperatures across SPEC2000 workloads. All temperatures are normalized to T_{max}, the peak temperature seen across all benchmarks and modules (83°C).

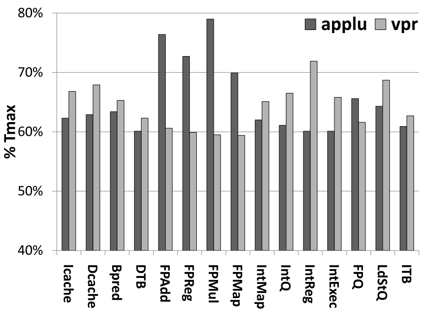
2.1 Workload Variation

Figure 1 shows the range of temperatures experienced by different structures within an Alpha21364-like processor [1] across a set of 8 SPECINT (*bzip2, gcc, gzip, mcf, perlbmk, twolf, vortex, vpr*) and 9 SPECFP benchmarks (*ammp, applu, apsi, art, equake, galgel, lucas, sixtrack, swim, wupwise*). All temperatures are normalized to the peak temperature, T_{max}, seen across all modules and benchmarks, which corresponds to the temperature of the `FPAdd` module when running *lucas* (83°C). Notice the significant variation in temperature within nearly every module. Apart from the more than 40% variation seen in `FPAdd` (a 37°C swing), other structures (whose utilizations are not as strongly correlated with the execution of floating-point and integer benchmarks) also exhibit significant temperature shifts, 10-15% for `Bpred` and `IntReg`. These large temperature ranges suggest that scheduling alone can be a powerful tool for manipulating aging rates.

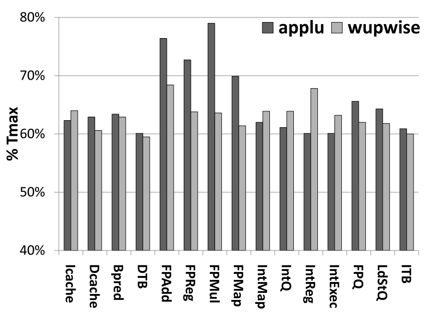
Figure 2 selects a few representative applications and examines them in greater detail. Figures 2(a) and 2(b) highlight how the traditional view of "hot" and "cold" applications is perhaps too simplistic. Without accounting for the module-level variation in temperatures, one could incorrectly assume that *applu* is more taxing, from a reliability perspective, than *vpr* or *wupwise* simply because it exhibits a higher peak operating temperature (`FPMul`). However, this would neglect the fact that for many structures, like `IntReg`, temperatures for *applu* are actually much lower than the other two applications. For completeness, Figure 2(c) is included to show that variations in module temperatures exist even between applications with comparable peak temperatures. All things considered, deciding where on the CMP to schedule a particular application, to achieve the least reliability impact, requires additional information about the strength of individual structures within every core. Although the magnitude of the temperature differences may not seem impressive at first, with peak deltas in module temperatures around 10-20% in Figure 2(a), these modest variations in temperature can have dramatic impacts on a processor's mean time to failure (MTTF).

2.2 Implications for Mean Time to Failure

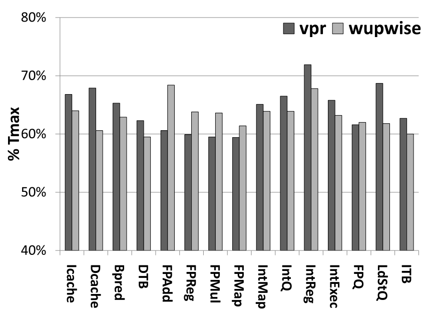
From Figure 2, one could expect a core consistently running *applu* to fail because of a fault in the `FPMul` unit due to its high operating temperatures. However, in the presence of process variation other structures within the core could have been manufactured with more defects (or tighter timing margins), and therefore even more susceptible to failure despite not ever realizing the same peak temperatures as `FPMul`. In this environment, a reliability-centric job scheduler must take into consideration the extent of damage present within a core in addition to the per-module thermal footprint of running applications. Figure 3 presents the expected lifetime of a core running *applu* or *vpr* as a function of the module identified as the



(a) SPECFP v. SPECINT



(b) SPECFP v. SPECFP



(c) Variation despite comparable peak temperatures

Figure 2: Head-to-head comparisons of applu (SPECFP), vpr (SPECINT), and wupwise (SPECFP). No one benchmark in (a), (b), or (c) strictly dominates the other (with respect to temperature) across all modules.

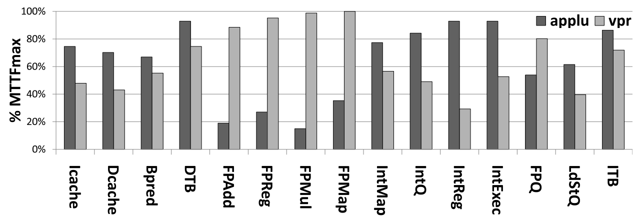


Figure 3: Projected core lifetime based on execution of *applu* and *vpr* as a function of the module identified as the weakest structure. Values are normalized to the best achievable MTTF.

weakest structure. The lifetimes are projected based on well-known MTTF equations for NBTI and TDDB [15, 21]. The values are normalized to the best achievable MTTF, which in this comparison is attained if `FPMap` is the weakest module in the core and the core is running *vpr*. The optimal job to schedule on a particular core to maximize its lifetime is dependent not just on the application mix currently available, but also on the strengths of individual structures within that core. Scheduling *applu* on a core with a weak `IntReg` can nearly triple its operating lifetime compared to naively forcing it to run *vpr*. Similarly, scheduling *vpr* instead of *applu* on a core with a weak `FPAdd` improves its projected lifetime by more than 4x.

To further highlight the need to address process and workload variation, a quick examination of the processors simulated in Section 4.1 reveals that 35% of core failures are the result of failing structures that never experience peak on-chip temperatures. Furthermore, 22% of core failures are caused by modules that do not rank among the top three most thermally active. By accounting for the impact of process variation and module-level thermal variation of applications, Maestro can prevent premature core failures and reap the opportunity left on the table by previous schedulers.

3. MAESTRO

Figure 4 presents a block diagram of Maestro, which consists of two main components: 1) a health monitoring system (introspection) and 2) a virtualization layer that implements wearout-centric job scheduling (management). Although this paper targets reliability-centric scheduling, a broader vision of introspective reliability management could use online sensor feedback to guide a range of solutions from traditional DVFS to more radical approaches like system-level reconfiguration [14].

3.1 Health Monitoring

Tracking the evolution of wearout damage within a CMP (i.e., health monitoring) is essential to forming intelligent reliability-centric schedules. Maestro assumes that the underlying CMP is provisioned with circuit-level sensors like those described in [22]. Recognizing that the two mechanisms addressed in this work, NBTI and TDDB, both impact physical device parameters as they evolve has led researchers to actively develop circuit-level sensors that can track these changes. NBTI is known to shift threshold voltage (V_t) leading to slower devices and increased subthreshold/standby leakage current (I_{ddq}), while TDDB increases gate currents (I_{gs} and I_{gd}). Both result in statistically measurable degradation in timing paths at the microarchitectural-level [3, 6].

A runtime system collects raw data streams from the array of circuit-level sensors and applies statistical filtering and trend analysis (similar to what is described in [3]) to convert these streams into descriptions of system characteristics including, delay profiles, leakage currents, and operating temperatures. These individual channels of information are then processed to generate a comprehensive microarchitectural-level reliability assessment of the CMP. This is shown in Figure 4 as a vector of per-module damage values (relative to the maximum damage sustainable prior to failure). Introducing the additional analysis step allows the health monitoring system to account for things like the presence of redundant devices within a structure, the influence of shifting environmental conditions on sensor readings, and the interaction between different wearout mechanisms. Ultimately, this allows the low-level sensor feedback to be abstracted with each vector representing the effective damage profile for a particular core.

3.2 Maestro Virtualization Layer

The second portion of the Maestro framework resides in system firmware that serves as the interface between the OS and the underlying hardware. The OS provides the virtualization layer with a set of jobs that need to run on the CMP and other meta-data (optional) that can guide Maestro in refining its scheduling policies (Section 3.2.3). Online profiling of system workloads identifies application-specific thermal footprints, shown in Figure 4 as a vector of per-module temperatures for each application. This thermal footprint can either be generated by brief exploratory execution of jobs on the available cores, similar to what is done in [26], or projected by correlating thermal behavior with program phases (leveraging the existing body of work on runtime phase monitoring and prediction). Given the prevalence of on-chip temperature sensors [13], Maestro assumes low-overhead exploration is performed

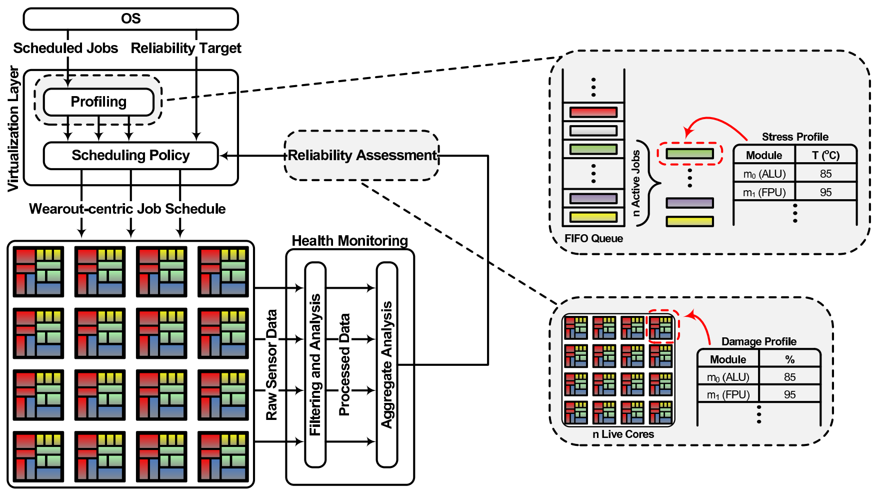


Figure 4: A high-level block diagram of the Maestro introspective reliability management system. Dynamic monitoring of sensor feedback and detailed characterization of workload behavior enables Maestro to improve lifetime system reliability with wearout-centric scheduling.

during each scheduling interval. Coupled with the real-time health assessments, this detailed module-level application characterization enables Maestro to create wearout-centric job schedules that intelligently manage CMP aging.

As previously defined, scheduling in this paper will refer to the act of mapping threads to cores and is initiated by two main events, 1) the OS issues new jobs for Maestro to execute (pushes into a FIFO queue) or 2) the damage profile of the underlying CMP has changed sufficiently (taking on the order of days/weeks) to warrant thread migration. The two reliability-centric scheduling policies evaluated in this work illustrate two approaches to lifetime reliability. The greedy policy (Section 3.2.2) takes the position that all core failures are unacceptable and aggressively preserves even the weakest cores. The adaptive policy (Section 3.2.3) champions a more unconventional philosophy that claims individual core failures are tolerable provided the lifetime reliability of the CMP system is maximized.

Both wearout-centric policies, and the naive baseline scheduler, are presented below along with corresponding pseudocode. Unless otherwise indicated, the following definitions are common to all policies: m, a microarchitectural module (i.e., `FPMul`, `IntReg`, etc.); $LiveCores$, the set of functional cores in the CMP, $\{c_0, c_1, ..., c_N\}$; $JobQueue$, the set of **all** pending, uncompleted jobs issued from the OS; $ActiveJobs$, the set of the N oldest, uncompleted, jobs, $\{j_0, j_1, ..., j_N\}$; $Dmg(m)$, the entry in the CMP damage profile for module m; $Temp(j, m)$, the entry for module m in the temperature footprint for job j.

3.2.1 *Naive Scheduler*

A standard round-robin scheduler is used as the baseline policy. The least-recently-used (LRU) core in the set of $LiveCores$ is assigned the oldest job from the set of $ActiveJobs$. This process is repeated until all jobs in $ActiveJobs$ have been scheduled. This policy maintains high-level load balancing by distributing jobs uniformly across the cores. However, without accounting for core damage profiles or application thermal footprints, the resulting schedule is effectively a random mapping (from a reliability perspective).

Algorithm 1: Greedy wearout-centric scheduler

```
Step 1:
    foreach $c \in LiveCores$ do
        find $c_{dmg}$ , the damage present in core $c$ , where
            $c_{dmg} \longleftarrow Dmg(m') \mid m' \in c \wedge Dmg(m') \geq Dmg(m), \forall m \in c$
    end
    sort $LiveCores$ based on $c_{dmg}$
end
Step 2:
    until $ActiveJobs$ is empty
        $c_w \longleftarrow$ weakest core in $LiveCores$ based on $c_{dmg}$
        $m_w \longleftarrow m' \mid m' \in c_w \wedge Dmg(m') \geq Dmg(m), \forall m \in c_w$
        foreach $j \in ActiveJobs$ do
            find $cost_{j,c_w}$ , the cost of executing job $j$ on core $c_w$ , where
                $cost_{j,c_w} \longleftarrow Temp(j, m_w)$
        end
        $j_{opt} \longleftarrow j' \mid j' \in ActiveJobs \wedge cost_{j',c_w} \leq cost_{j,c_w}, \forall j \in ActiveJobs$
        Assign job $j_{opt}$ to core $c_w$
        Remove $c_w$ from $LiveCores$ and $j_{opt}$ from $ActiveJobs$
    end
end
```

3.2.2 *Greedy Scheduler*

This policy attempts to minimize the number of premature core failures by greedily favoring the weakest cores (Algorithm 1). Cores are sorted based upon their damage profiles and priority is given to the cores whose weakest modules possess the most damage (Step 1 of Algorithm 1). These "weak" cores are greedily assigned jobs with the most favorable thermal footprints with respect to their damage profiles (Step 2 of Algorithm 1), minimizing their effective thermal stress. This *local wear-leveling* reduces the probability that these weak cores will fail due to a *single* damaged structure. Scheduling the weak cores first maximizes the probability of finding jobs with favorable thermal footprints with respect to each weak core since there is a larger application mix to choose from. However, this also forces the stronger cores to execute the remaining, potentially less desirable, jobs. In practice, this means that

Algorithm 2: Adaptive wearout-centric scheduler

```
let $GA(J, C)$ be the optimal schedule generated by the GA for jobs $J$ and cores $C$
Step 1:
    foreach $c \in LiveCores$ do
        find $c_{dmg}$ , the damage present in core $c$ , where
            $c_{dmg} \longleftarrow \sum_{m_i}^{c} \alpha_i Dmg(m_i)$ and $\alpha_i$ is a scaling factor
            biased toward modules with more damage
    end
    sort $LiveCores$ in increasing order of $c_{dmg}$
    $PrimaryCores \longleftarrow$ first $n$ cores where $n$ is set by the user through the OS
    $SecondaryCores \longleftarrow$ remaining $N - n$ cores
end
Step 2:
    let $S_{primary}$, be the set of job-to-core assignments, $(j, c)$, $\forall c \in PrimaryCores$
    $S_{primary} \longleftarrow GA(ActiveJobs, PrimaryCores)$
    Assign jobs for $PrimaryCores$ according to $S_{primary}$
    Remove assigned jobs from $ActiveJobs$
end
Step 3:
    let $S_{secondary}$, be the set of job-to-core assignments, $(j, c)$, $\forall c \in SecondaryCores$
    $S_{secondary} \longleftarrow GA(ActiveJobs, SecondaryCores)$
    Assign jobs for $SecondaryCores$ according to $S_{secondary}$
end
```

the stronger cores in the CMP actually sacrifice a portion of their lifetime to lighten the burden on their weaker counterparts (*global wear-leveling*).

3.2.3 Adaptive Scheduler

The adaptive scheduler recognizes that many CMP systems are often underutilized, provisioned with more cores than they typically have jobs to run (see Section 4.3). The scheduler exploits this fact by allowing a few weak cores to be sacrificed in order to preserve the remaining stronger cores (Algorithm 2). Although being complicit in core failures may seem non-intuitive, in systems that are underutilized, the greedy scheduler can lead to CMPs that are overprovisioned early in the CMP's life ($LiveCores >> JobQueue$) while not assuring enough available throughput ($LiveCores < JobQueue$) later on. This insight forms the basis of the adaptive policy.

Promoting a survival-of-the-fittest environment, this policy maximizes the functional life of the strongest subset of cores ($PrimaryCores$ in Step 1 of Algorithm 2), those with the least amount of initial damage and the potential to have the longest lifetimes. By assigning jobs to the $PrimaryCores$ first, Maestro ensures that they execute applications with the most appropriate thermal footprints (Step 2 of Algorithm 2). The remaining jobs are assigned amongst the $SecondaryCores$ (Step 3 of Algorithm 2). This can lead to some weak cores failing sooner than under a greedy policy. Note, however, in Step 3 of Algorithm 2, the scheduler is still looking amongst the remaining jobs for the one with the best thermal footprint given a core's damage profile. This *local wear-leveling*, common to both the greedy and adaptive policies, ensures that the weaker cores even under the adaptive policy survive longer than they would under the naive policy. Ultimately, over the lifetime of the CMP, if $PrimaryCores \geq JobQueue$ consistently, while avoiding periods when $PrimaryCores >> JobQueue$ or $PrimaryCores < JobQueue$, then Maestro has maximized the total amount of computation performed by the system. The proper size of $PrimaryCores$, n, is exposed to the OS so that the behavior of the scheduler can be customized to the needs of the end user.

Finally, note in Step 2 and Step 3 of Algorithm 2, the scheduler uses an optimization scheme based on a genetic algorithm (GA) to identify the least-cost schedules for both the $PrimaryCores$ and $SecondaryCores$. This allows the adaptive scheduler to consider the effect scheduling a job has on all structures within a core (unlike the greedy scheduler which only looks at the weakest structure) for more effective *local wear-leveling*. The optimization used in this work is derived from [8], a standard solution of the generalized assignment problem, and is described below [1].

Chromosome definition: The chromosome modeled is a job-to-core mapping of a set of n jobs, $J = \{j_0, j_1, ..., j_n\}$, to a set of m cores $C = \{c_0, c_1, ..., c_n\}$. It is represented as a one-dimensional array where the value stored at index i, j_i, is the job that has been assigned to core i. The example in Figure 5 has jobs j_1 mapped to core 0, j_{n-1} mapped to core 1, and j_0 mapped to core m. During **Step 2** of the adaptive scheduling algorithm $n > m$, while for the optimization performed in **Step 3** $m = n$.

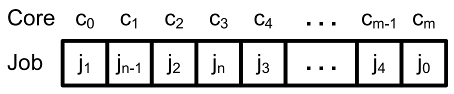


Figure 5: Chromosome structure

Cost function: The cost function used by the GA is recalculated at each scheduling interval, based on the CMP damage profile and application thermal footprints, according to Equation 1, where $Cost(S)$ = the cost of schedule S and $Cost(j, c)$ = the cost of scheduling job j on core c.

$$\begin{aligned} Cost(S) &= \sum_{j,c}^{S} Cost(j, c) \\ &= \sum_{j,c}^{S} \Big(\sum_{m}^{c} Dmg(m) \cdot Temp(j, m) \Big) \end{aligned} \quad (1)$$

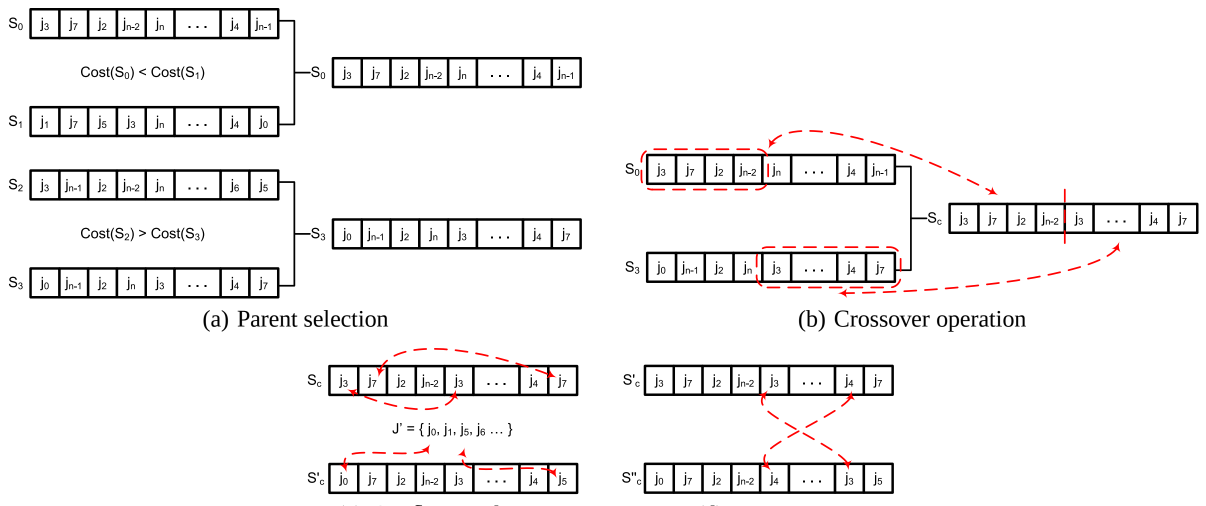
The individual steps of the GA are enumerated below:

1. **Generate initial population:** An initial population of solutions (schedules) is created by randomly enumerating a subset of the possible job-to-core mappings.

2. **Evaluate fitness:** Calculate the fitness (cost) of all members of the population using Equation 1.

3. **Reproduction:** Two parents are identified, each using a simple binary tournament where two candidates are selected randomly from the population and the one with the best fitness (smallest cost) is chosen for reproduction (Figure 6(a)). A child is generated by applying a one-point crossover operator on the parent chromosomes, where a random crossover point $i \in [0, m]$ is selected, where m is the size of the chromosomes. The child chromosome is formed by combining the first i genes from one parent with the last $m - i$ genes from the second parent (Figure 6(b)) . Note that this newly formed chromosome could have the same job assigned to two different cores. For this case to arise there must also be a set of jobs $J' \subset J$ that are unassigned since $n \geq m$. To resolve the

[1]The runtime overhead of the GA is negligible for long-running scientific and server workloads. However, for shorter-running applications the GA optimization can be replaced by a greedy version without severely impacting the effectiveness of the adaptive scheduler.



(a) Parent selection (b) Crossover operation (c) Conflict resolution (d) Mutation

Figure 6: Steps involved in reproduction. S_0, S_1, S_2, S_3 are the parental candidates. S_c is the resulting child chromosome after initial crossover. S'_c and S''_c are the states of the child chromosome after conflicts resolution and mutation respectively.

conflicts, one of the redundant cores (selected at random) is reassigned a job from J' based on $Cost(j, c)$ (Figure 6(c)). Lastly, the newly formed child chromosome is mutated by taking 2 randomly selected job assignments and swapping them (no risk of creating new conflicts), reducing the probability of converging at local optima (Figure 6(d)) .

4. **Replace and Repeat:** After a child solution is formed the weakest member, as defined by the cost function, of the existing population is replaced by the new child. This concludes a single generation in the evolutionary cycle. The process is repeated until a predetermined number of generations, gen_{max}, fails to produce an improved solution. [2]

4. EVALUATION AND ANALYSIS

This section evaluates Maestro's reliability-centric scheduling policies using lifetime reliability simulations. A variety of system parameters including CMP size and system utilization are varied to investigate their impact on Maestro's performance. The effectiveness of each wearout-centric policy is measured in terms of *lifetime throughput* (LT), the number of cycles spent executing active jobs (real applications not idle threads), summed across all cores, throughout the entire lifetime of the CMP. LT improvement metrics are the result of comparisons with the naive, round-robin scheduler presented in Section 3.2.1. Monte Carlo experiments are conducted using a simulation setup similar to the framework in [12]. The standard toolchain of SimAlpha, Wattch [5], and Hotspot [20] is used to simulate the thermal characteristics of workloads and Varius [19] is used to model the impact of process variation. Results presented in this section, unless otherwise indicated, are for a 16-core CMP with processors modeled after the DEC Alpha 21264/21364 [1].

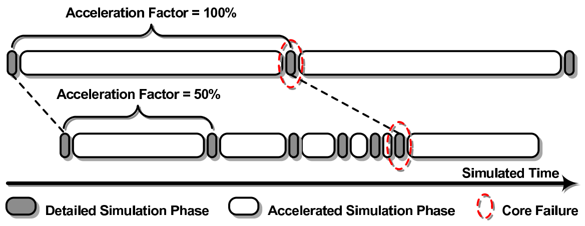
Given that CMPs have lifespans on the order of years (3-5 years in future computer systems [11]), detailed lifetime reliability simulation is a computationally intensive task. This is especially true when large numbers of Monte Carlo runs must be conducted to generate statistically significant results. Since wearout damage takes years to reach critical mass, results presented in this section were gathered using an *adaptive* simulation scheme. Short periods of detailed system-level reliability simulation, the darker phases in Figure 7(a), are used to gather statistics on the progression of CMP aging in light of dynamically changing workload streams and Maestro's reliability-centric scheduling. The simulation is then rapidly advanced through a longer period of time (accelerated simulation) using the statistics generated during the most recent detailed phase as a guide. To minimize error, the length of the accelerated simulation phase is limited by the amount of damage accumulated during the detailed interval according to Equation 2:

$$L_a = (\frac{D_{fail}}{D_{acc}}) \cdot AF \cdot L_d \qquad (2)$$

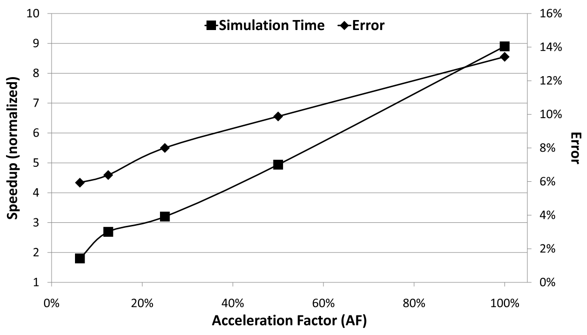
where, L_a = length of the accelerated phase, L_d = length of the previous detailed phase, D_{fail} = amount of damage the weakest core in the CMP can sustain before failing, D_{acc} = amount of damage accumulated by the weakest core during the previous detailed phase, and AF = parameter that trades off simulation time for accuracy (0%-100%).

Dynamically adjusting the durations allows simulation to slow down as cores near their failing point, where small changes in damage and scheduling decisions have larger implications. When a core fails in phase i, accelerated simulation resumes at a faster rate ($L_{a_{i+1}} > L_{a_i}$), but L_a soon contracts as the next core in the CMP nears failure. Figure 7(a) illustrates (not to scale) how adjusting AF can influence the lengths of the accelerated phases. The value of AF essentially dictates the number of detailed phases that are simulated between core failures. At an AF of 100%, simulations are accelerated from one core failure to the next. However, when AF is dialed down to 50%, many more phases are required to cover the same amount of simulated time, concentrating simulation effort around times when cores are failing and improving simulation accuracy. Figure 7(b) shows both simulation time speedup and error as a function of AF, illustrating how simulation time can be traded off for fidelity. The experiments presented in this work use an AF of 6%, resulting in simulation runtimes from 30 minutes to over 6 hours for a single set of Monte Carlo runs.

[2]Given the size of the solution space, as many as 16! possible schedules for a 16-core CMP, values of gen_{max} from 0 to 100,000 were studied to understand the tradeoff between optimality and runtime. The actual values of gen_{max} used in Section 4 were determined empirically based on the CMP size, with many runs producing good results with gen_{max} as low as 1000.


Acceleration Factor = 100%
Acceleration Factor = 50%
Simulated Time
Detailed Simulation Phase
Accelerated Simulation Phase
Core Failure

(a) Interleaving of detailed and accelerated simulation phases.



(b) Simulation time/error v. acceleration factor (AF).

Figure 7: The adaptive simulation used to accelerate lifetime reliability simulations while incurring minimal experimental

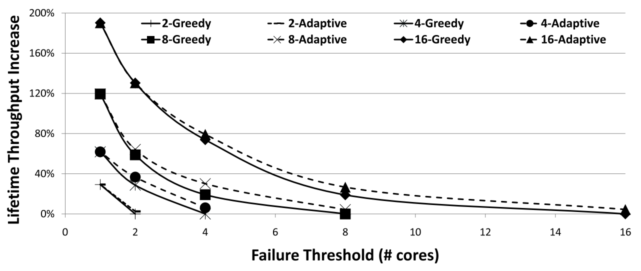
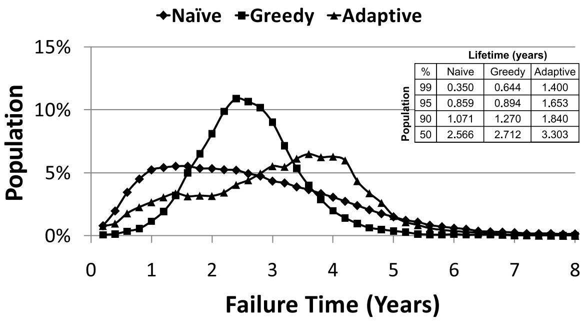


Figure 8: Performance of wearout-centric scheduling policies verses CMP size and failure threshold.

4.1 Lifetime Throughput Enhancement

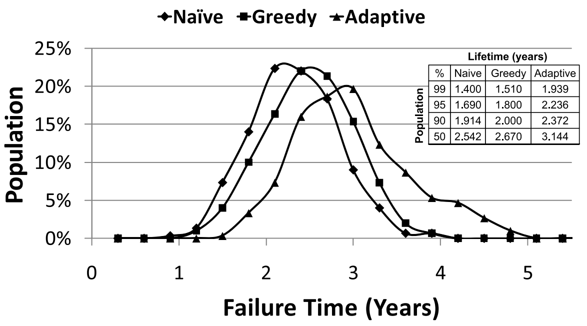
Figure 8 shows the normalized LT improvement as a function of the scheduling policy, CMP size, and failure threshold. In the context of this paper, failure threshold is defined as the number of cores that must fail before a chip is considered unusable. This is the point at which the risks/costs associated with maintaining a system with only a fraction of its original computational capacity justifies replacing the chip. The CMP is considered dead even though functional cores still remain. The results shown in Figure 8 are conducted for 2 to 16-core systems, and failure thresholds ranging from 1 core to all cores. The value of the failure threshold is passed to the adaptive policy so that it can optimize for the appropriate number of cores. Results are shown for CMP utilizations of 100%, providing a lower-bound on the benefits of the adaptive policy (Section 4.3 examines the impact of CMP utilization).

As expected, both the greedy and adaptive policies perform well across all CMP sizes and the majority of failure thresholds. As the size of the CMP grows, Maestro has more cores to work with, increasing the chances of finding complementary job-to-core mappings. This results in more effective schedules for both wearout-centric policies improving their performance. Yet even with the



		Lifetime (years)	
%	Naive	Greedy	Adaptive
99	0.350	0.644	1.400
95	0.859	0.894	1.653
90	1.071	1.270	1.840
50	2.566	2.712	3.303

(a) Failure distribution (Core)



		Lifetime (years)	
%	Naive	Greedy	Adaptive
99	1.400	1.510	1.939
95	1.690	1.800	2.236
90	1.914	2.000	2.372
50	2.542	2.670	3.144

(b) CMP failure distribution (CMP)

Figure 9: Failure distributions for individual cores and the 16-core CMP with a failure threshold of 8 cores and 100% utilization. Trendlines are added (between markers) to improve readability.

lack of scheduling alternatives in a 2-core system, both policies can still achieve a respectable 30% improvement.

A strong dependence on failure threshold is also evident. By aggressively minimizing premature core failures, the greedy scheduler achieves large gains for small failure thresholds. However, as the failure threshold nears the size of the CMP, the LT improvement attenuates. This is expected since under the greedy policy, stronger cores sacrifice a portion of their lifetime in order to preserve their weaker counterparts. The cost of this sacrifice is most apparent when the failure threshold allows all the cores to fail. In these systems, the increased contribution toward LT by the weak cores is offset by the loss in LT resulting from the strong cores failing earlier. Notice also that the adaptive scheduler outperforms greedy by the largest margins when the failure threshold is roughly half the size of the CMP. In these situations, the adaptive scheduler has the maximum freedom to sacrifice $SecondaryCores$ to preserve $PrimaryCores$ (Section 3.2.3). At either extreme for failure threshold, it performs similarly to greedy.

Lastly, it is important to note that, although the benefits of wearout-centric scheduling are less impressive for these extreme values of failure threshold, the scenarios when a user could actually afford to wait for all the cores within a system to fail are also quite remote. For the remainder of the paper, all the experiments shown are for a 16-core CMP with a failure threshold of 8 cores and 100% system utilization unless otherwise indicated.

4.2 Failure Distributions

Figure 9 presents the failure distributions for the individual cores, as well as the CMPs that correspond to the results in Figure 8. Figure 9(a) illustrates the effectiveness of the wearout-centric policies at distributing the workload stress appropriately. The distri-

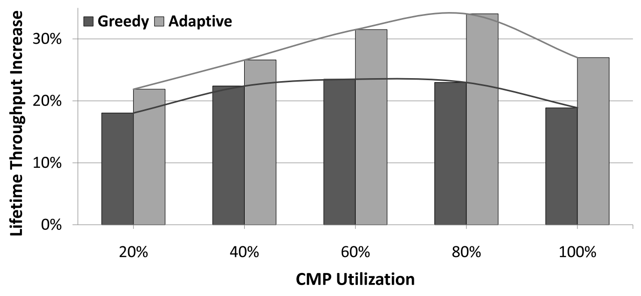


Figure 10: Impact of CMP utilization on reliability enhancement.

bution for the baseline naive policy reveals a bias towards early premature core failures. The greedy scheduler, exploiting effective wear-leveling, produced a tighter distribution, lacking in both premature failures as well as cores that significantly outlasted their peers. Lastly, the adaptive policy also delivers on its promises by preserving a subset of cores for a longer period of time than either the naive or greedy schedulers.

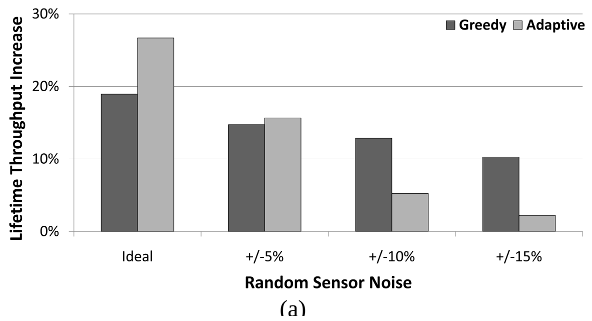
Figure 9(b) tells a similar story, but with chip-level failures. As with the individual core distributions, both wearout-centric policies are able to increase the mean failure time of the CMP population. Note that because the failure time of a CMP is limited by the weakest set of its constituent cores, the distributions in Figure 9(b) are considerably tighter than those in Figure 9(a). The corresponding tables of expected lifetimes embedded within the plots present the data slightly differently. From a product yield/warranty perspective, intelligent wearout-centric scheduling can be thought of as an additional means of ensuring that cores meet their expected reliability qualified lifetimes. For example, the table in Figure 9(b) shows that the adaptive scheduler enabled 99% of the chips to survive beyond 1.9 years, compared to just 1.4 years with the naive baseline, a 38% improvement. Granted, job assignment alone cannot make *guarantees* on lifetime, but it can complement existing more aggressive techniques like thermal throttling.

4.3 Sensitivity to System Utilization

The utilization of computer systems can be highly variable, both within the same domain (e.g., variability inside data centers) and across domains. One might expect computationally intensive scientific codes (e.g., physics simulations, oil exploration, etc.) to consistently utilize the hardware. On the other hand, since designers build web servers to accommodate peak loads (periodic by season, day, and hour), they are often over-provisioned for the common case. Some reports claim average utilization as low as 20% of peak [2].

Figure 10 plots the performance of Maestro's wearout-centric schedulers as a function of system utilization. The results are shown for nominal utilizations ranging from 20% (light duty mail server or embedded system) to 100% (scientific cluster)[3]. Note that initially as average utilization drops, improvement in lifetime throughput actually increases. A system that is slightly underutilized can be more aggressively load balanced since some cores are allowed to remain idle. However, as utilization continues to drop these gains are eventually lost, until finally improvements are actually worse than at full utilization. In these highly over-provisioned systems, the efforts of wearout-centric scheduling to prevent premature failures are *partially* wasted because so few cores are actually necessary to sustain demand. Nevertheless, in the long run, the periodic spikes in utilization do accumulate, and thanks to the longer overall core lifetimes (lower utilization means less overall stress that translates to longer lifetimes), the greedy and adaptive schedulers still manage to exhibit improvements.

[3]Although the mean utilization per simulation run is fixed, the instantaneous utilization experienced by the CMP is allowed to vary over time, sometimes peaking at 100% even for a system nominally at 20% load. Furthermore, the average *effective* utilization is also changing as cores on the CMP begin to fail.



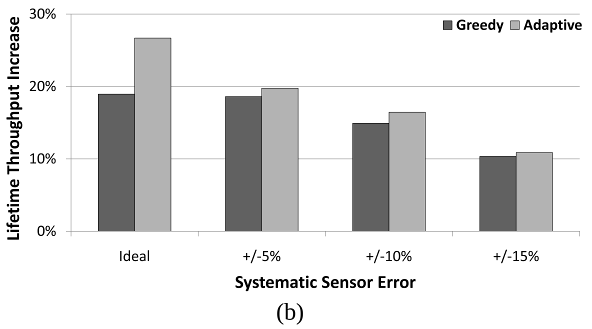


Figure 11: Sensitivity to sensor noise. Although random sensor noise can be removed with the appropriate filtering, systematic error due to manufacturing tolerances is more problematic.

4.4 Sensitivity to Sensor Noise

Figure 11 illustrates how error-prone sensors could impact lifetime reliability gains. Although the introduction of systematic error, which is studied in Figure 11(b), does reduce the potential of wearout-centric scheduling, the presence of random noise (more common for circuit-level sensors) shown in Figure 11(a) can be accounted for and mitigated by the statistical filtering and trend analysis schemes referenced in Section 3.1. Yet, even at the extreme of +/-15% systematic error, Maestro still achieves over 10% LT improvement. Figure 11(b) also suggests that the adaptive scheduler is more sensitive to noise than the greedy scheduler. By aggressively trying to preserve $PrimaryCores$, the adaptive heuristic relies strongly on sensor feedback to accurately identify the boundary between its two classes of processors, making it less robust against sensor inaccuracy.

4.5 Sensor Selection

Lastly, Figure 12 presents a comparison between the low-level damage sensors advocated in this work and more conventional hardware like temperature sensors and performance counters. Given that Maestro is targeting an environment with significant amounts of process variation, it is not surprising that employing temperature and activity readings as proxies for wearout/manufacturing induced damage is inadequate. They are unable to account for the extent to

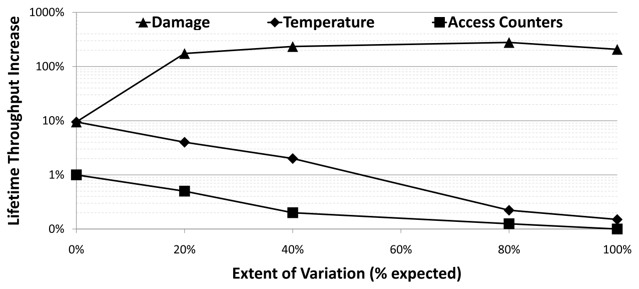


Figure 12: Performance of wearout-centric scheduling with different sensors. Results are shown for a failure threshold of 1 core to favor the temperature sensor and access counter based approaches.

which non-uniform, pre-existing damage within the CMP responds to the same thermal stimuli. In the absence of variation, a scheduler relying on only temperature might effectively enhance lifetime reliability by evenly distributing the thermal stress across the CMP. However, without any knowledge of CMP damage profiles, as process variation is swept from one extreme (no variation) to the other (100% expected variation at 32nm), thermal load balancing alone is insufficient and Figure 12 shows a dramatic plunge in the effectiveness of these temperature based schemes. Similarly, the performance counter approach performed poorly across the spectrum of variation.

5. CONCLUSION

As large CMP systems grow in popularity and technology scaling continues to exacerbate lifetime reliability challenges, the research community must develop innovative ways for systems to dynamically adapt. Although issues like process variation are the source of design and validation nightmares, this inherent heterogeneity in future systems is also a source of potential opportunity. Maestro recognizes that although emerging reliability obstacles cannot be ignored, with the appropriate monitoring and intelligent management, they can be overcome. By exploiting low-level sensor feedback, Maestro was able to demonstrate the effectiveness of wearout-centric scheduling at preventing premature core failures, improving expected CMP lifetimes by as much as 38%. Formulating wearout-centric schedules that achieved both local and global wear-leveling, Maestro enhanced the lifetime throughput of a 16-core CMP by as much as 180%. Future work that leverages sensor feedback to improve upon other traditional reliability management mechanisms (e.g., DVFS) could demonstrate still more potential.

6. REFERENCES

[1] Alpha. 21364 family, 2001. http://www.alphaprocessors.com/21364.htm.

[2] A. Andrzejak, M. Arlitt, and J. Rolia. Bounding the resource savings of utility computing models, Dec. 2002. HP Laboratories, http://www.hpl.hp.com/techreports/2002/HPL-2002-339.html.

[3] J. Blome, S. Feng, S. Gupta, and S. Mahlke. Self-calibrating online wearout detection. In *Proc. of the 40th Annual International Symposium on Microarchitecture*, pages 109–120, 2007.

[4] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[5] D. Brooks, V. Tiwari, and M. Martonosi. A framework for architectural-level power analysis and optimizations. In *Proc. of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.

[6] A. Cabe, Z. Qi, S. Wooters, T. Blalock, and M. Stan. Small embeddable nbti sensors (sens) for tracking on-chip performance decay. Washington, DC, USA, Mar. 2009. IEEE Computer Society.

[7] J. Choi, C. Cher, , H. Franke, H. Haman, A. Wedger, and P. Bose. Thermal-aware task scheduling at the system software level. In *Proc. of the 2007 International Symposium on Low Power Electronics and Design*, pages 213–218, Aug. 2007.

[8] P. C. Chu and J. E. Beasley. A genetic algorithm for the generalised assignment problem. 24(1):17–23, 1997.

[9] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *Proc. of the 33rd Annual International Symposium on Computer Architecture*, June 2006.

[10] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner. Razor: Circuit-level correction of timing errors for low-power operation. In *Proc. of the 37th Annual International Symposium on Microarchitecture*, pages 10–20, 2004.

[11] C. Evangs-Pughe. Live fast, die young [nanometer-scale ic life expectancy]. *IEE Review*, 50(7):34–37, 2004.

[12] S. Feng, S. Gupta, and S. Mahlke. Olay: Combat the signs of aging with intropsective reliability management. In *Proc. of the Workshop on Architectural Reliability*, June 2008.

[13] J. Friedrich et al. Desing of the power6 microprocessor, Feb. 2007. In *Proc. of ISSCC*.

[14] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke. The stagenet fabric for constructing resilient multicore systems. In *Proc. of the 41st Annual International Symposium on Microarchitecture*, pages 141–151, 2008.

[15] X. Li, B. Huang, J. Qin, X. Zhang, M. Talmor, Z. Gur, and J. B. Bernstein. Deep submicron cmos integrated circuit reliability simulation with spice. In *Proc. of the 2005 International Symposium on Quality of Electronic Design*, pages 382–389, Mar. 2005.

[16] Z. Lu, J. Lach, M. R. Stan, and K. Skadron. Improved thermal management with reliability banking. *IEEE Micro*, 25(6):40–49, Nov. 2005.

[17] M. Powell, M. Gomaa, and T. Vijaykumar. Heat-and-run: Leveraging smt and cmp to manage power density through the operating system. In *12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 260–270, Oct. 2004.

[18] D. Roberts, R. Dreslinski, E. Karl, T. Mudge, D. Sylvester, and D. Blaauw. When homogeneous becomes heterogeneous: Wearout aware task scheduling for streaming applications. In *Proc. of the Workshop on Operationg System Support for Heterogeneous Multicore Architectures*, Sept. 2007.

[19] S. 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*, pages 3–13, Feb. 2008.

[20] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Transactions on Architecture and Code Optimization*, 1(1):94–125, 2004.

[21] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The case for lifetime reliability-aware microprocessors. In *Proc. of the 31st Annual International Symposium on Computer Architecture*, pages 276–287, June 2004.

[22] D. Sylvester, D. Blaauw, and E. Karl. Elastic: An adaptive self-healing architecture for unpredictable silicon. *IEEE Journal of Design and Test*, 23(6):484–490, 2006.

[23] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *Proc. of the 35th Annual International Symposium on Computer Architecture*, pages 363–374, June 2008.

[24] A. Tiwari, S. Sarangi, and J. Torrellas. Recycle: Pipeline adaptation to tolerate process variation. In *Proc. of the 34th Annual International Symposium on Computer Architecture*, pages 323–334, June 2007.

[25] A. Tiwari and J. Torrellas. Facelift: Hiding and slowing down aging in multicores. In *Proc. of the 41st Annual International Symposium on Microarchitecture*, pages 129–140, Dec. 2008.

[26] J. Winter and D. Albonesi. Scheduling algorithms for unpredictably heterogeneous cmp architectures. In *Proc. of the 2008 International Conference on Dependable Systems and Networks*, page To appear, June 2008.