Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip Multiprocessors

Ayse K. Coskun, Richard Strong, Dean M. Tullsen, and Tajana Simunic Rosing
Computer Science and Engineering—University of California, San Diego
{acoskun, rstrong, tullsen, tajana}@cs.ucsd.edu

ABSTRACT

Temperature-induced reliability issues are among the major challenges for multicore architectures. Thermal hot spots and thermal cycles combine to degrade reliability. This research presents new reliability-aware job scheduling and power management approaches for chip multiprocessors. Accurate evaluation of these policies requires a novel simulation framework that can capture architecture-level effects over tens of seconds or longer, while also capturing thermal interactions among cores resulting from dynamic scheduling policies. Using this framework and a set of new thermal management policies, this work shows that techniques that offer similar performance, energy, and even peak temperature can differ significantly in their effects on the expected processor lifetime.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques

General Terms

Reliability

1. INTRODUCTION

The microprocessor industry has moved to chip multiprocessing to enable the scaling of performance beyond the limits of uniprocessor execution. As the chip area shrinks, the power density grows for new process technologies, causing higher temperatures. Therefore, our success at finding ways to profitably use the available transistors has a cost, as we are now faced with significant challenges in managing the power and thermal effects on these chips. High temperatures increase the cost of cooling, degrade reliability, and reduce performance.

A number of mechanisms for thermal control are currently available for multicore processors—including job scheduling, job migration, dynamic voltage and frequency scaling, etc. This paper presents a framework for evaluating the effectiveness of these techniques in various combinations, and presents effective new policies for managing thermal effects.

The primary goal of managing temperature is to prevent processor failure. This research is focused on hard failures (which cause permanent damage to the underlying circuit), and the physical and electrical phenomena that induce them. Silicon devices have a number of failure modes that are impacted by temperature, and in some cases these modes are at odds: thermal management techniques that reduce the rate of one failure mode may exacerbate another. In fact, we show that techniques that are nearly identical in performance, power, and even peak temperature can differ by a factor of two in expected processor lifetime. Therefore, it is critical that we have a model of power, temperature, and particularly reliability that incorporates all critical failure modes. This paper introduces such an integrated modeling framework, shows that some policies have unintended consequences when all sources of failure are considered, and proposes new policies that provide significant gains in processor lifetime with little loss in performance.

Most power and thermal management techniques have focused primarily on controlling peak temperature. Although several types of failures clearly scale with the peak temperature, that factor alone does not accurately model all types of failures. Other failures, such as cracks and fatigue failures, are created not by sustained high temperatures, but rather by the repeated heating and cooling of sections of the processor. This phenomenon is referred to as thermal cycling. The particular failures that our infrastructure models include electromigration, time dependent dielectric breakdown, and the thermal cycling-induced errors mentioned above. Failing to include thermal cycling in the failure model can lead to misleading results and highly suboptimal temperature mitigation strategies.

Modeling thermal cycles is difficult. The primary challenge is the need to accurately model these systems over the timescales which thermal cycles occur. This far exceeds the ability of current processor modeling techniques, which typically simulate system behavior at instruction or cycle level. Therefore, we introduce new performance modeling mechanisms, integrated with our power, thermal, and reliability models, that allows accurate modeling of execution behavior over tens or hundreds of seconds. Being able to capture all the thermal failure effects is critical to an accurate understanding of processor lifetime. We show, for example, that some proposed mechanisms that appear to improve reliability if thermal cycling effects are ignored actually have the opposite effect when thermal cycling is taken into account.

In this work, we define several new scheduling and power management policies. This research shows that the most critical factors for increasing processor lifetime with acceptable performance are: (1) The asymmetric thermal characteristics of the cores (cores in the center having very different properties than those on the edges, etc.); (2) The frequency of migration, which can both inhibit sleep states and cause thermal cycling. Our most effective policy that employs voltage/frequency scaling, as well as our best one that does not, both account for the location asymmetry and reduce the num-

SIGMETRICS/Performance'09, June 15–19, 2009, Seattle, WA, USA.

ber of thread movements. We present new scheduling policies that can decrease the failure rate by a factor of two (over naive management), with a performance cost of less than 4%.

This paper is organized as follows. Section 2 discusses recent work in thermal and reliability management. Section 3 describes the integrated performance, power, thermal, and reliability simulation framework. We provide the details of the experimental methodology in Section 4, and explain the thermal management techniques in Section 5. A thorough evaluation of the techniques is presented in Section 6, and we conclude in Section 7.

2. BACKGROUND AND RELATED WORK

In this section we provide an overview of previously proposed dynamic thermal management (DTM) techniques, and also simulation methodologies for modeling performance, power and reliability. Many of the DTM techniques are reactive in nature—depending on sensors to indicate temperatures beyond assigned thresholds, and adapting the processor to either reduce or migrate activity to bring the temperature down.

Brooks and Martonosi [4] introduced the concept of dynamic thermal management in reaction to thermal measurements. Some thermal management techniques stall execution or migrate computation to other units to control temperature. A well-known example of such techniques is clock gating, which freezes all dynamic operations until the thermal emergency is over, causing typically significant performance cost. Clock gating is implemented in Intel's Pentium 4 [13].

Donald et al. [10] propose a dynamic thermal management technique for simultaneous multithreading (SMT) architectures. Their technique selectively manages the execution of integer or floating point intensive threads to prevent hot spots in the register files. To identify integer or floating point intensive threads, hardware event counters are sampled during execution. In activity migration, the heat is spread by moving computation to a different location on the die. Migration can happen at multiple levels: from one core to another [14], or within a core [28].

Existing redundancy in a superscalar pipeline can be utilized for controlling temperature [24]. In this technique, the power density is controlled by balancing the utilization of issue queues, register files and ALUs. Fetch gating alternates between fetching and stalling fetch in order to reduce the activity and power density in the pipeline. Skadron et al. introduce a feedback control loop to control the duty cycle for fetch-gating [28].

Another class of thermal management techniques use dynamic voltage and frequency scaling (DVFS, where the system is able to alter the processor's frequency and supply voltage dynamically) to respond to thermal emergencies [28]. DVFS can use different number of steps for the global voltage and frequency settings, ranging from two in Intel's SpeedStep technology to 40 for the Intel XScale.

In hybrid thermal management [27], for mild levels of thermal stress fetch gating (where we stall fetch, but allow other stages of the pipeline to proceed with previously fetched instructions) is used as the response mechanism. When the overhead of the fetch gating increases and instruction level parallelism (ILP) cannot sufficiently hide the ill performance effect, DTM switches to DVFS. Another hybrid DTM technique minimizes the performance impact by proactive use of software techniques like thermal-aware process scheduling combined with reactive use of hardware techniques such as clock gating [19]. Donald, et al. evaluate various combinations of DVFS, clock gating (i.e., stop-go) and migration for managing the temperature of multicore processors [11]. They show that distributed DVFS combined with thread migration provides the best performance among different alternatives.

Prior work has also investigated low overhead temperature-aware task scheduling at the operating system level for multiprocessor system-on-chips (MPSoC) [8], and the authors proposed an adaptive probabilistic policy addressing both temperature variations and hot spots. The adaptive scheduling technique is combined with thread migration or DVFS to further improve thermal behavior. Murali, et al. [22] propose a technique that assigns frequencies to different cores in an MPSoC to guarantee meeting the thermal constraints. In the offline phase, the set of feasible frequencies for different temperature and workload constraints are calculated by solving convex optimization models. At run time, the management policy selects the appropriate frequency values that meet the current workloads and operating conditions.

In Heat-and-Run [12], the authors propose a DTM solution for chip multiprocessors with SMT cores. SMT thread assignment is used to maximize processor resource utilization by co-scheduling threads that use complementary resources before cooling is necessary. In this way the cost of thread migration is reduced.

Few papers in the thermal management literature have taken reliability explicitly into account. Reliability management has been mostly addressed previously as a way of optimizing the policies or architecture at design-time. The Reliability-Aware Microprocessor (RAMP) provides a reliability model at the architecture level for temperature related intrinsic hard failures [30]. It analyzes the effects of application behavior on reliability and optimizes the architectural configuration and the voltage/frequency setting statically (at design time) to meet the reliability target. Previous work also shows that aggressive power management can adversely affect reliability due to fast thermal cycles, and optimization methods that consider reliability constraints can provide energy savings while improving the MPSoC lifetime [25].

To the best of our knowledge, a simulation framework to evaluate the reliability impact of dynamic management policies in a fast and accurate way has not been introduced previously. The SimPoint tool [26] also addresses the problem of long simulation times, but it provides clustering analysis to identify a few representative points that can be simulated to predict the performance of the entire application. Instead, we want to capture the entire behavior rather than summarize. However, we use SimPoint's phase identification mechanism to capture a complete phase trace as part of our simulation process. Biesbrouck, et al. [1] use individual program phase information (a complete phase trace not unlike ours) to guide multithreaded simulation. This is accomplished by creating a *Co-Phase Matrix*, which represents the per-thread performance for each potential combination of the single-threaded phase behaviors that occur when multiple programs are run together. Although RAMP [30] also integrates an architecture-level performance simulator with a power model and a thermal simulator, it does not include the phase-based approach we introduced. Using our framework, we are able to simulate much longer periods of real-life execution in reasonable simulation time.

3. A NOVEL FRAMEWORK FOR MULTI-CORE RELIABILITY MODELING

This area of research presents new methodological challenges that require tools and solutions radically different than traditional architectural investigation. This section describes the entire simulation infrastructure, but with a focus on the two most novel aspects of the framework, which are the long time-frame performance modeling and the integrated reliability model.

3.1 Overview

For a study such as this one, it is critical that we have a fully

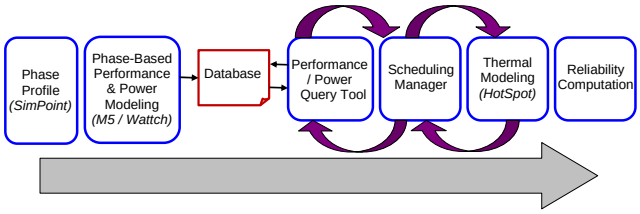

Figure 1: Design Flow

integrated performance, power, and thermal model of the entire chip multiprocessor. This is because we are modeling interactive scheduling techniques that observe the temperature and possibly power characteristics of the processor and make scheduling decisions accordingly. For example, it is impossible to completely decouple the performance and the thermal models. However, full interactive architecture-level simulation is also not possible, as just a single simulation at these time frames corresponding to several minutes of real execution time could require months to complete.

Our simulation framework is shown in Figure 1. The performance modeling front-end combines a full-program phase profile combined with detailed architecture-level simulation of every distinct program phase at all possible frequency settings, including both performance and power characteristics. This characterization all goes into a database that can be queried as the full CMP simulation progresses. In this way, we can model the effects of changing frequency, stopping or migrating jobs, etc., without further architecture-level simulation. After scheduling decisions are made and the resulting performance and power data produced, we can model time-varying temperature effects across the entire chip. The temperature curves are then fed into the reliability models, producing the expected failure rates.

3.2 Long-Term Performance Modeling

To accurately model the reliability of the system including thermal cycling effects, we need to capture temporal thermal behavior over time periods orders of magnitude longer than typically modeled in architectural simulation. At the same time, we would like to capture various types of effects that the architectural simulation provides; e.g., workload-dependent utilization of specific architectural structures and their impact on power and temperature, the time-varying behavior of individual applications, etc. This requires the development of new simulation tools and methodologies not currently available.

We initially use SimPoint [26] to capture the phases of each application. But instead of capturing one or a few representative phases, we use it to capture a complete phase profile of each application, beginning to end. Then, using the M5 performance simulator [2] integrated with the Wattch power modeling tool [5] and utilizing a finite number of simulation samples for each phase, we can reconstruct the power and execution properties of the complete program. In fact, we do this for all voltage and frequency settings available, so that we can reconstruct the complete program even in the face of an arbitrary number of voltage/frequency changes.

We capture these program traces in a database which can be queried by the Schedule Manager at distinct intervals. Given a program start point, an interval length in cycles, and a frequency setting, the query tool returns the average instructions per second (IPC) and power levels across the interval, and the point in execution the program reaches at the end of the interval. Thus, at runtime, the scheduling manager can make decisions about thread migration, thread stoppage, or voltage/frequency changes, and query the database to model the precise effects.

This framework relies on two simplifying assumptions that are critical to making this problem tractable. The first is that the time constants over which temperature varies do not require us to fully capture cycle-by-cycle variances in the temperature portion of the model. The instruction-level variations are captured in the performance model, but only summarized in the latter stages. This allows us to replace the cycle-by-cycle data with a stair-step graph, presenting performance and power behavior as constant at the average values over individual intervals. This way, we capture the program behavior with little loss of accuracy.

The second assumption is that the behavior of individual threads is separable. This is accurate because we model systems with private L2 caches, which is a likely architectural scenario in future systems [21]. At 16 cores and above, the interconnect cost of a shared cache would be extremely high. This assumption has been used and demonstrated to be accurate even on research that does not require this type of long simulation [20]. Even with the small core counts in current multicores, the AMD dual-core and quad-core Opteron, the IBM Power6, and the coming Intel Nehalem processor all have private L1 and L2 caches. For shared L2 caches, interactions between threads will be higher and system-level accuracy will be reduced. However, recent research on multicore caching has focused on reducing those interactions [16], and in the extreme, the proposed techniques could be configured to make the shared caches essentially act as private caches. Thus, even in that scenario we can represent a reasonable system accurately. This assumption makes it difficult to model parallel applications with any significant communication between threads, if that communication impacts the runtime characteristics of the application.

3.2.1 Phase Modeling

We used SimPoint [26] to identify the various phases within the applications and to characterize complete program execution. A program's execution is divided into intervals of 100 million instructions. Once we assign each interval to a representative phase, we represent a program's execution by a Phase-ID trace [1]. Thus, at any instruction during a program's execution, we use this file to determine the current phase and to identify points of transition between phases.

By running simulations at each phase point in M5 and composing performance/power statistics with the Phase-ID trace, we create both a power and a performance trace. The scheduling manager then accesses these traces via the query tool.

3.2.2 Power Modeling and Management

Power modeling requires coupling the execution traces obtained from M5 with a tool that computes the power consumption for each functional unit. This coupling converts the performance parameters (e.g., cache accesses, branch predictions, etc.) into estimates for transistor switching, and then the power model utilizes these estimates for calculating the instantaneous power values.

Transistors consume power when they switch output values, but they also leak power even when they do not switch. The former is referred to as dynamic power, and historically has been the dominant factor; however, as technology shrinks, leakage power becomes increasingly important. We utilized Wattch [5] for the dynamic power modeling of cores in our framework. We integrated Wattch with M5 to provide dynamic and cycle accurate power measurements for each application. To model power dissipation of L2 caches, we used CACTI [34] (an integrated memory performance, area, leakage, and dynamic power model) and obtained the typical power consumption of a memory block with the given size and properties, and then used these values throughout the simulation.

We developed a power model for 65nm by scaling the parameters within Wattch to match published power values for 65nm technology. The variation in dynamic power range we observed matches the power distribution on a similar core [15], on which the majority of applications had less than 16% power dissipation difference from the other applications. Among the applications sampled in that distribution were the SPEC suite, which we use in this study.

We compute the leakage power of CPU cores based on structure areas, temperature, and supply voltage. For the 65nm process technology, we assume a leakage power density of $0.5W/mm^2$ at 383K [3]. To account for the temperature and voltage, we used the second-order polynomial model proposed by Su, et al. [33]. This model computes the change in leakage power for the given differential temperature and voltage values. We determined the coefficients in the polynomial model empirically to match the normalized leakage values in the paper. This model is found to match closely with measurements [33], and we found the leakage values produced were in line with expected values (i.e., 30–40% of the total power consumption) based on the technology.

One of the techniques we investigate to manage power is Dynamic Power Management (DPM). DPM puts cores in sleep state to save energy. We implemented a fixed timeout policy [18], which is one of the commonly used DPM policies. For each core, the policy waits for a timeout period when the core is idle, and then turns off the core. This is to ensure that we do not turn off cores for very short idle times, where turning off the core would not amortize the cost of transitioning to and from the sleep state. The time period to amortize the cost of going to sleep is called the breakeven time (t_{be}). We assume a sleep state power value of 0.05W, and based on the active and idle power dissipation values we computed the t_{be} to be around 200ms. A simple and effective way to set the timeout period is $t_{timeout} = t_{be}$ [18].

3.3 Thread Management and Thermal Modeling

We implemented a scheduling manager which enables the simulation of a large array of thread management policies. The mechanisms available for managing temperature include adjusting the frequency/voltage of a core (DVFS), putting an idle core into a low-power sleep mode (DPM), migrating computation off of a hot core, and policies that stop activity on a hot core (i.e., clock- or fetch-gating). We present the specific policies we model in Section 5. In each policy, the scheduling manager makes a set of decisions after each scheduling interval, and it may incorporate performance and thermal information from the prior interval. After making those decisions for each thread and core, the scheduling manager then queries the performance database to obtain the power and performance behavior of each core over the next interval. Our simulation sampling intervals (50 ms) are shorter than a scheduling interval, so there would be multiple exchanges with the performance database before another scheduling decision is made.

Since the scheduling manager keeps track of performance and power information, it also has the responsibility of modeling complex phenomena such as the delay from thread migrations. The model simulates the effects on power and performance for the following phenomena: thread migration, DVFS, starting a new application on a core, core sleep, and core wakeup. The assumptions we made for several of the delays modeled are presented in Table 1, but one of the more complex phenomena deserves special attention.

We modeled two aspects of the cost of thread migration among cores. We measured the software overhead in M5's full system mode as the time for Linux to migrate a thread from one core to another idle core and to start execution. This thread migration takes less than 3.0 μs. We also attributed architecture overhead to cold start effects in the branch predictor, caches, TLBs, etc. We measured cold start effects by inducing many random migrations for each benchmark and computing the average loss in performance. The average loss was 204 μs, but varied wildly by benchmark—i.e., from 2 to 740 μs. Note that cold start effects dominate the migration penalty. To address the highly variable overhead, we modeled a distinct migration penalty for each benchmark.

Parameter	Model Value
Sampling Interval	50ms
Thread Migration Delay	syscall delay + cold start effects
DVFS Delay	syscall delay + 20E-6s
Wakeup Delay	25ms
Application Startup Delay	syscall delay + cold start effects
Transition Power (to and from sleep states)	10W

Table 1: Delay and Power Model Assumptions

Parameter	Value
Die Thickness	$0.1mm$
Core Area	$14.44mm^2$
L2 area (total of 2 banks)	$10.56mm^2$
Convection Capacitance	140 J/K
Convection Resistance	0.1 K/W

Table 2: HotSpot Parameters

Automated thermal modeling requires power traces for each unit as input, in addition to the chip and package characteristics such as die thickness, heat sink convection properties, etc. Therefore, we feed the detailed power trace derived by the combination of the scheduling manager and the performance/power database into the thermal model. We modified HotSpot Version 4.0's [28] (block model) settings to model the thermal characteristics of the 16-core die. We used the steady state temperature of each unit as the initial temperature values. We summarize the HotSpot parameters in Table 2. We calculated the die characteristics based on the trends reported for 65nm process technology [14].

The described methodology allows us to do full-program simulation with simple lookups of sampled simulation data. This sacrifices some accuracy. However, the rate at which temperature changes typically dwarfs the time of even complete phases, so we would expect this technique to actually sacrifice little accuracy. We validated our methodology by comparing the results with direct M5/ Wattch power output. For each phase simulation point of each SPEC benchmark, we ran M5 and Wattch for 500ms of simulated execution and gathered power statistics every $500\mu s$. We compared the power statistics of M5/Wattch and our framework, and we found that our phase-based approach has 1.8% error overall. Table 3 shows the detailed results for the benchmarks. *bzip2* with input set *program* had the largest average error of 3.0%.

The low error margin in our power computation methodology translates to even lower error in temperature computation because of the thermal time constants. To verify the accuracy of our methodology in terms of the temperature response, we experimented with bzip2, as it has the highest power error margin. Figure 2 shows one particular (worst case) data point—the temperature trace for a core running bzip2 and then going to sleep state, on a system running 12 bzip2 threads. The "M5/Wattch" thermal trace corresponds to the detailed power trace sampled at $500\mu s$, and the "phase" trace is the thermal output of running the same workload and using our power computation methodology. We observe that the trace generated with our methodology closely matches the trace sampled at a

Benchmark	Average Error	Benchmark	Average Error
parser	0.023	facerec	0.022
applu	0.021	gcc_166	0.020
art110	0.016	fma3d	0.024
swim	0.018	mcf	0.011
galgel	0.015	gap	0.020
twolf	0.009	vpr_route	0.017
mesa	0.027	ammp	0.015
lucas	0.009	bzip2_program	0.030
vortex1	0.018	equake	0.029
sixtrack	0.011	eon_rushmeier	0.018
apsi	0.014	crafty	0.018
Overall: 0.018			

Table 3: Power Estimation Error of Our Front-End Tool Compared with respect to M5/Wattch

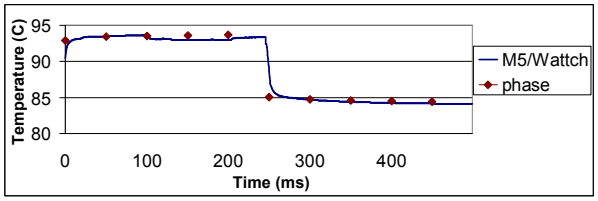

Figure 2: Comparison of Temperature Responses for *bzip2*, Using Two Simulation Methodologies.

higher granularity. As bzip2 is one of the most power-variant applications, the rest of the benchmarks demonstrate even less difference. Because thermal cycling effects are insignificant unless the temperature variations are more than a few degrees, these results are more than accurate enough to capture both temperature-induced and cycle-induced effects.

Once we generate a full thermal trace, we use this trace as input to our reliability model described in the next section.

3.4 Reliability Modeling

Our work targets temperature-induced reliability problems. Our simulation and modeling framework allows us to evaluate scheduling policies based on their success in reducing the failure rate due to thermal hot spots and thermal cycles. Achieving a lower failure rate increases the mean-time-to-failure, which is the expected lifetime of the circuit. The most commonly studied temperature-induced intrinsic hard failure mechanisms are electromigration, time dependent dielectric breakdown, and thermal cycling [17, 30].

Electromigration (EM) occurs in interconnects as a result of the momentum transfer from electrons to ions that construct the interconnect lattice and leads to hard failures such as opens and shorts in metal lines. The EM failure rate (λ_{EM}), based on Black's model, is given in Equation 1. In the equation, E_a is the activation energy, k is the Boltzmann's constant, T is the temperature, J and J_{crit} are the current density and the threshold current density, respectively, and A_0 is a material dependent constant. We represent the first half of the equation with the term λ^0_{EM}, which can be considered as a constant (an average technology/circuit dependent value).

$$\lambda_{EM} = A_0(J - J_{crit})^{-n} e^{(-E_a/kT)} = \lambda^0_{EM} e^{(-E_a/kT)} \quad (1)$$

Time dependent dielectric breakdown (TDDB) is a wear-out mechanism of the gate dielectric, and failure occurs when a conductive path is formed in the dielectric. TDDB is caused by the electric field and temperature, and the failure rate is defined in Equation 2. Similar to the EM failure rate equation, we use λ^0_{TDDB} to represent the first half of the equation. Both EM and TDDB failure rates are exponentially dependent on temperature.

$$\lambda_{TDDB} = A_0 e^{\gamma E_{ox}} e^{(-E_a/kT)} = \lambda^0_{TDDB} e^{(-E_a/kT)} \quad (2)$$

Thermal cycling (TC) is caused by the large difference in thermal expansion coefficients of metallic and dielectric materials, and leads to cracks and other permanent failures. The thermal cycling effect is modeled by the Coffin-Mason equation [17]. Slow thermal cycles happen because of low frequency power changes such as power on/off cycles. Fast cycles occur due to events such as power management decisions. Although lower frequency cycles have generally received more attention, recent work shows that thermal cycles due to power or workload variations can also degrade reliability [23, 25]. The failure rate due to thermal cycling is formulated as in Equation 3.

$$\lambda_{TC} = C_0(\Delta T - \Delta T_o)^{-q} f \quad (3)$$

In this equation, ΔT is the temperature cycling range. The elastic portion of the thermal cycle is shown as ΔT_o. Elastic thermal stress refers to reversible deformation occurring during a cycle, and ΔT_o should be subtracted from the total strain range. Typically, $\Delta T_o << \Delta T$ [17], so the ΔT_o component can be dropped from the equation. C_0 is a material dependent constant, q is the Coffin-Manson exponent, and f is the frequency of thermal cycles. Note that the Coffin-Manson equation [17] computes the number of cycles to failure. Therefore, the MTTF (in years) is the number of cycles multiplied by the period of the cycles.

Computing the frequency of cycles is not straightforward in a simulation of an irregular, dynamic system. To resolve this problem, we observed the recent temperature history on each core to compute ΔT and f. We set the initial length of the history window to 5 seconds, and adjusted the length dynamically depending on how many cycles were observed. For example, if no temperature cycles were observed in the last interval, we incremented the history window length to capture slower cycles. ΔT is the temperature differential we observed in the last interval. We set a higher band of 80% and a lower band of 20% of the temperature range recorded in the last interval, and counted the number of times the temperature exceeded the higher band or went below the lower band, and used that to calculate the number of cycles in this period. In this way, we could account for the contribution of cycles with varying temperature differentials and varying periods.

To combine the failure rates, we used the sum-of-failure-rates model as in RAMP [30]. This model assumes that all the individual failure rates are independent. Mean-time-to-failure (MTTF) is $1/\lambda$ for constant failure rates; therefore, we averaged the failure rate observed throughout the simulation and computed the corresponding average MTTF. The average MTTF value reported for 65nm technology is 7 years [32].

For moderate temperatures at 65nm technology, Srinivasan, et al. [31] demonstrate that the contribution of electromigration, dielectric breakdown, and cycling to the overall failure rate are similar to each other. This allows us to calibrate the constants in each failure equation (λ^0_{EM}, λ^0_{TDDB}, and C_0) to give a system MTTF of 7 years at nominal temperature. We used the same constants all throughout the experiments, which means that the relative impact of different failure mechanisms might change depending on the conditions. For example, if the temperature is high, then the effect of EM or TDDB is higher than TC.

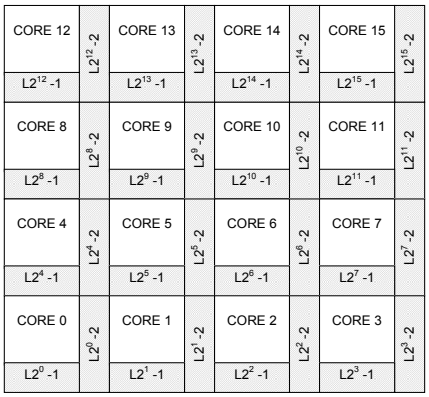

Figure 3: Floorplan of the 16-core CPU

CPU Clock	2.0Ghz
ICache	64KB 2-way @1ns (2 cyc)
DCache	64KB 2-way @1ns (2 cyc)
L2Cache	2MB 8-way @10ns (20 cyc) (2 banks)
Memory Latency	100ns (200 cyc)
Branch Predictor	21264-style tournament predictor
Issue	out-of-order
ROB	128 entry
Issue Width	4
Functional Units	4 IntAlu, 2 IntMult, 2 FPALU, 2 FPMultDiv
Physical Regs	128 Int, 128 FP
IQ entries	64
Vdd	1.2V
DVFS Settings	100%, 95%, 85%

Table 4: Architectural Parameters

We also examined thermal gradients, which refer to the temperature differences among different locations on the die. However, we do not explicitly include the effects of gradients in our overall reliability model. This is because although thermal gradients can induce hard errors, their primary effect is on device latencies, which are then manifested as an increase in timing errors, rather than hard failures.

4. METHODOLOGY

This section describes other details of our simulation infrastructure that impact the results shown in the following sections. These, in general, are details that are relatively independent of our framework described in Section 3 and easily changed, such as the processor core model, the workload, etc.

The M5 Simulator's [2] out-of-order execution model is based on SimpleScalar 3.0 [6], and provides a detailed model of an Alpha 21264 processor. Anticipating continued scaling of core counts, the CPU we model is a 16-core multiprocessor manufactured at 65nm. The floorplan for this CPU is provided in Figure 3. Each core has out-of-order issue, a private data cache, instruction cache, L2 cache, and memory channels. Each core possesses three voltage and frequency settings for dynamic voltage/frequency scaling: 1.200V at 2.0GHz, 1.187V at 1.900GHz, and 1.06V at 1.7GHz which represent DVFS settings of 100% (original), 95% (step-1), and 85% (step-2), respectively. The architectural parameters of each core are depicted in Table 4.

Creating representative workloads is a challenge in a 16-core environment. To assist this process, we classify all SPEC2K benchmarks in terms of their variability and memory boundedness (discussed below). The distinction between CPU bound and memory bound applications is particularly important in this study because it impacts how performance scales as the frequency changes. We model both homogeneous and heterogeneous workloads in terms of the applications' CPU or memory boundedness. As our execution model does not extend to parallel programs, the homogeneous workloads stand in for both homogeneous server-type workloads and parallel applications with few stalls for communication. However, our homogeneous and heterogeneous multiprogrammed workloads best represent a server environment, where the average lifetime of the processor can significantly affect overall costs.

We use the ratio of memory-bus transactions to instructions as a metric to classify applications as memory or CPU-bound, as suggested by Wu et al. [35]. We classify applications along several other dimensions. By constructing our workloads from applications with different phase variability, power savings potential and CPU/memory boundedness, we seek to represent a wide range of real world workloads.

Table 5 describes each workload. We model workloads with 12–16 threads—our CMP architecture is constructed to not have thermal issues when lightly loaded, which is the expected behavior for the next few processor generations. We construct both homogeneous and heterogeneous workloads, and CPU-bound, memory-bound, and mixed workloads. The mixed workloads contain applications from both extremes, as well as some in the middle of our categorization. In the time frames we model, several of the applications complete execution. In those cases, we continually restart the application at the beginning to get consistent behavior across the experiment.

A common performance metric on multicore platforms is a raw count of IPC. However, this metric gives undeserved bias towards high-IPC threads as performance may be increased by running more CPU bound threads. To circumvent this difficulty, we used the Fair Speedup Metric (FS) [7, 29]. FS is computed by finding the harmonic mean of each thread's speed-up over a baseline policy of running the thread at the highest frequency and voltage.

Although some applications complete multiple times during our simulations, we compute FS in such a way that the overall contribution of each application is the same.

5. RELIABILITY-AWARE SCHEDULING

The simulation and modeling infrastructure described allows us to design and evaluate several job allocation and thermal management strategies. We divide these techniques into three categories, those that change what is running on a core (via gating or migration), those that continue to execute the same thread but change speed (via DVFS), and hybrids that combine the two types.

Each of these methods can be integrated with Dynamic Power Management (DPM) as well. DPM turns off cores after they have been idle for a given timeout period. The scheduling and thermal management policies evaluate the system characteristics at every scheduling period, and make a decision accordingly. In all cases, the scheduling tick is set to every 200ms. The threshold temperature for all the temperature-triggered policies is 85^{o}C. The **default** policy keeps the initial assignment of jobs to cores fixed, and no workload migration or voltage/frequency scaling occurs on the fly.

5.1 Migration and Gating Scheduling Policies

These techniques attempt to move computation off of hot cores, either via migration or stalled execution as a response to a thermal event (high temperature) or as a matter of policy.

Stop_Go [11] runs each core at the default (highest) frequency and voltage setting until a core reaches the thermal threshold. At this point, the core is stalled and the clock is gated to reduce power

Wkload name	Description	Cores Utilized	Benchmarks
hom_16_cpu	Homogeneous CPU Bound	16	sixtrack*16
hom_16_mem	Homogeneous MEM Bound	16	mcf*16
het_16_cpu	Heterogeneous CPU Bound	16	mesa*3, bzip2_program*3, crafty*2, eon_rushmeier*3, vortex1*2, sixtrack*3
het_16_mem	Heterogeneous MEM Bound	16	mcf*4, art110*4, equake*3, gcc_166*3, swim*2
het_16_mix	Heterogeneous MIX	16	mcf*2, mesa, art110, sixtrack*2, equake, bzip2_program, eon_rushmeier*2 swim, applu, twolf, crafty, apsi, lucas
het_12_cpu	Heterogeneous CPU Bound	12	mesa*2, bzip2_program*3, crafty*2, eon_rushmeier*2, vortex1*1, sixtrack*2
het_14_cpu	Heterogeneous CPU Bound	14	mesa*2, bzip2_program*3, crafty*2, eon_rushmeier*2, vortex1*2, sixtrack*3
het_12_mix	Heterogeneous MIX	12	mcf*2, mesa, art110, sixtrack*2, eon_rushmeier*2, swim, crafty, apsi, lucas
het_14_mix	Heterogeneous MIX	14	mcf*2, mesa, art110, sixtrack*2, equake, eon_rushmeier*2, swim, twolf, crafty, apsi, lucas

Table 5: Workload Characteristics

J_3	J_{10}	J_7	J_2
J_6	J_{14}	J_{15}	J_{11}
J_{12}	J_{16}	J_{13}	J_5
J_1	J_8	J_9	J_4

Figure 4: Thread Assignment Strategy for Balance Location

consumption. If the core's temperature goes below the temperature threshold in the next sampling interval, execution continues. We assume that each core can be clock-gated individually.

Migration sends jobs that have exceeded a thermal threshold to the coolest core that has not been assigned a new thread during the current scheduling period. If the coolest core selected is already running a job, we swap the jobs among the hot and cool cores. This technique can be thought of as an extension of core-hopping or activity migration techniques [12, 14] to the case of many cores and many threads.

Balance assigns jobs with the highest committed IPC during the last interval (i.e., between the last two scheduling ticks) to cores that have the lowest temperature. This scheduling idea represents a more proactive form of migration in which threads are dynamically assigned to locations before thermal thresholds necessitate action.

Balance_Location is similar to balance, but instead of assigning the threads with the highest committed IPC to the coolest cores, it assigns them to cores that are expected to be coolest based on location. The cores on the corner locations of the floorplan are expected to be the coolest; the remaining cores on the sides are expected to be the second coolest; and the cores in the center of the floorplan are hottest. This is because the temperature of a core is a result not only of activity on that core, but also on the activity of its neighbors: higher number of active neighbors results in hotter cores. Figure 4 shows the strategy we used to assign 16 jobs (j_1 to j_{16}) to cores, where the jobs have decreasing committed IPC ($IPC_1 > IPC_2 > ... > IPC_{16}$). Whereas the optimal allocation of threads to cores might diverge from the allocation shown in the figure depending on the IPC difference among threads, this allocation generally results in temperature characteristics close to the best allocation. With this scheme and 14 threads, for example, cores labeled j_{15} and j_{16} in this figure would always be idle.

We also experimented with heuristics that choose a thread's next core allocation based on the temperature of the thread's current core (e.g., move the thread on the hottest to the coolest core), but these heuristics performed poorly. In multicore architectures like the one we model, location is a more significant factor than the execution characteristics of the threads in determining core temperature. Thus, those techniques ended up constantly moving jobs between hot and cold cores.

5.2 Methods with Voltage/Frequency Scaling

This set of techniques rely exclusively on dynamic voltage and frequency scaling to control thermal dynamics. They differ in how and when DVFS is applied.

DVFS-Threshold (dvfs_t) reduces voltage and frequency (V/f) one step at a time when a core's temperature exceeds a threshold. After reducing the V/f to the step-1 (95%) setting, if the core is still above the threshold in the next scheduling interval, dvfs_t uses the step-2 (85%) setting. When a core's temperature is below the threshold, the V/f setting is increased, again one step at each scheduling interval.

DVFS-location (location_dvfs) uses a fixed V/f setting for each core, and there is no dynamic scaling at runtime. As the center cores tend to heat up more quickly, the four cores in the center of the floorplan have the 85% setting. The corner cores are typically the coolest cores, hence they use the 100% (original) V/f setting. The rest of the cores (i.e., the eight remaining cores on the sides) have the 95% setting.

DVFS-Performance (dvfs_perf) reduces the voltage and frequency dynamically on a core depending on the memory boundedness of the current application phase. Previously, it was shown that CPU-intensive tasks do not gain much in terms of energy savings from running at low frequencies; and conversely, it is beneficial to run memory-bound tasks at a lower frequency [9], as their performance is much more tolerant of frequency scaling. DVFS-Performance seeks to reduce the overall chip temperature with minimal performance cost by proactively scaling back those applications that are least impacted.

To determine the memory-bound phases, we use a cycles-per-instruction (CPI) based metric, μ, as defined by Dhiman et al. [9]. It compares the observed CPI with a potential CPI we might have gotten without memory events. If the μ is near one, the application is CPU-bound. If it is low, the application is memory-bound. Note that μ can also take negative values. Analysis on our own application set confirms that this metric tracks extremely well with performance degradation in the presence of DVFS.

If the μ observed in the last interval is less than -0.8, then we use the 85% setting, and we find less than 6% performance loss during those phases. If $-0.8 < \mu < 0.5$, we apply the 95% setting, which induces less than 5% loss in performance. For $\mu > 0.5$, we do not perform any V/f scaling. When $\mu > 0.5$, if we used the 85% scaling for CPU-bound applications, the performance loss would be in the range of 12–15%.

DVFS-Performance_Threshold (dvfs_perf_t) behaves exactly

like *dvfs_perf* unless a core reaches a thermal threshold. If the temperature exceeds the threshold on a core, then the policy activates *dvfs_t* to reduce the temperature on that core. After the core's temperature returns within threshold, we switch back to *dvfs_perf*.

This technique wins if by proactively slowing a thread that is tolerant of frequency changes, it can enable a nearby thread that is not so tolerant of frequency change to forego a DVFS slowdown.

5.3 Techniques Combining Workload Allocation and DVFS

In investigating the interaction of scheduling and DVFS policies, we employ Balance_Location to represent the scheduling policies. It has useful properties in terms of both reliability and performance. It does only enough migration to find the best location for each thread, then only migrates when application characteristics change.

Balance_Location & DVFS-Threshold works by initially using *Balance_Location* to assign potentially hotter threads to cooler locations on the die. If this technique fails to keep a given core under the specified threshold, the core employs *dvfs_t* until it is under the thermal threshold.

Balance_Location & DVFS_Performance uses the *Balance_Location* policy to allocate jobs to cores, and runs *dvfs_perf_t* at the same time to decide on the V/f settings of the cores.

Balance_Location & DVFS-Location assigns the location V/f settings as in the *location_dvfs* policy to cores, and performs *Balance_Location* for allocating the threads. This tends to have the effect of assigning the most memory-bound threads in the center zone, which runs at the 85% setting.

Balance_Location uses IPC in assigning threads to locations. When combined with DVFS, we must account for the V/f and its effect on the measured IPC. Thus, if a core is running at a lower V/f setting, we scale the measured IPC based on the average performance hit observed at that V/f level.

6. EXPERIMENTAL RESULTS

In this section we demonstrate that the framework we proposed allows us to evaluate a large set of previously proposed and new scheduling algorithms, in terms of performance, power, temperature, and processor lifetime (reliability). We show that having a fully integrated model, including a reliability model that accounts for all the major causes of temperature-induced hard failures, sheds some new light on CMP scheduling.

Section 5 identified a wide assortment of thread management policies. In evaluating those policies in various execution scenarios, this section attempts to sort out the key issues facing the designer of a multicore thread management policy, such as: (1) how to properly combine scheduling/migration policies, DVFS policies, and DPM policies; (2) how to address peak temperature effects without exacerbating thermal cycling; (3) whether to use reactive or proactive DVFS policies; and (4) how to address thermal asymmetries in the chip multiprocessor.

We group the experiments in four major categories. Section 6.1 looks at full core utilization scenarios with a varying number of memory-bound and CPU-bound threads, using the five 16-thread workloads from Table 5. For these experiments, threads are initially placed on the cores randomly (i.e., with neither a clearly good or bad initial allocation). Section 6.2 examines systems that are less than fully utilized, with 12 or 14 jobs (i.e., 2 or 4 idle cores). Section 6.3 takes a deeper look at the consequences of the initial allocation of idle cores. Finally, Section 6.4 investigates how reliability, performance, and energy vary when the system has DPM capabilities, and which schedulers best complement DPM to achieve the desired trade-offs for reliability, energy, and performance.

To deliver a fair comparison, we present the energy and performance of the policies in addition to reliability (mean-time-to-failure). This is in lieu of trying to create a single artificial metric that captures all three: such user-defined metrics are susceptible to providing results that are specific to the assumptions made while creating the metric or while weighing the individual parameters.

We normalized all results in the following sections with respect to the *default* case of no thermal management (i.e., all threads running full speed on the initially assigned cores). Hence, the y-axes in our MTTF, performance, temperature and energy plots demonstrate the normalized values for these parameters. This allows us to evaluate each policy on the same scale.

Srinivasan et al. [32] reported the average MTTF of the SPEC suite simulated for 65nm at 1.0V of supply voltage as 7 years, and our model is calibrated to the same value. However, if the reliability model was re-calibrated to assume a shorter or longer MTTF, the policies would still display the same trends; only the absolute numbers would change depending on process technology, baseline MTTF, and the system being modeled. Therefore, we show results based on the % change in MTTF values, rather than absolute numbers, so that the dependence on the absolute calibration is minimal.

6.1 Full Core Utilization

Sections 6.1.1 to 6.1.3 examine the case where all cores are actively running threads.

6.1.1 Techniques Utilizing Workload Allocation

This section analyzes the workload allocation policies' ability to improve thermal characteristics. The policies that we analyzed in this section include *Stop_Go*, *Migration*, *Balance*, and *Balance_Location*. The results in Figure 5, which are the average values for all the 16-thread workloads, indicate that *Migration*, *Balance*, and *Balance_Location* have little impact on reliability in this scenario—this is because cores are fully utilized and most of our workloads are highly homogeneous.

In the one heterogeneous workload, the effect is still small. In that case, the *Balance* and *Balance_Location* policies each improve reliability by 4.4% with minimal impact (less than 1%) on performance, energy, and average temperature. Thus, in the absence of idle cores, these policies are less effective than the ones with voltage and frequency scaling, which we discuss in Section 6.1.2.

The *Stop_Go* policy was notably different than the policies discussed above, as it has the ability to cool a core even in the absence of idle cores. *Stop_Go* improved the MTTF by 65%, but with a hefty 52% decrease in performance and a 69% increase in energy consumption. Average temperature of the processor was reduced by 8%. The *Stop_Go* policy is prone to creating large temperature variations due to switching among active and idle states. However, the frequency of stalling/resuming execution was high enough that the temperature variations were of a relatively low magnitude, and the reliability of the core was dominated by the thermal hot spots only (i.e., no significant increase in cycling-based failures).

6.1.2 Techniques with Voltage/Frequency Scaling

Figure 6 shows the effect of the DVFS policies on reliability when the cores are fully utilized. DVFS has a much more significant impact than the workload allocation policies, due to its ability to reduce temperature even in the face of full utilization.

In particular, we find several key insights in these results. First, it is important to always keep an eye on peak temperature. *dvfs_perf*, by selectively choosing which threads to scale, sacrifices very little in performance, but does lag a bit behind in MTTF in comparison to other DVFS policies. This is because it ignores thermal warnings.

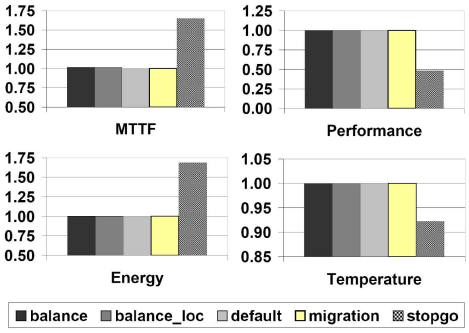

Figure 5: Comparison of Workload Allocation Techniques

dvfs_perf_t reacts upon reaching a threshold as well, and as a result loses some performance, but it has one of the lowest failure rates.

Second, we see significant benefits of proactive techniques over traditional reactive techniques. It is interesting to note that the other DVFS policies beat *dvfs_t* along all axes, which is particularly surprising on the performance front. This is surprising because (1) *dvfs_t* only scales when it *has* to, and (2) the other DVFS policies default to *dvfs_t* upon reaching the threshold temperature. The reason that other DVFS policies perform better is that proactively scaling a thread (whose performance is tolerant to scaling) reduces the temperature in that area, and often prevents other neighboring threads from reaching the threshold.

Third, we see that it is critical that our thread management policy understands the inherent thermal asymmetry of the multicore system (an asymmetry that will exist, in all likelihood, for any multicore greater than four cores). The policy that provides the best balance among all three metrics is *location_dvfs*, with a failure rate that is half of the baseline and a minimal performance loss (3.8% of default). To further investigate this point, we compared *location_dvfs* with homogeneous proactive scaling: all cores at *85% DVFS* and all cores at 95% DVFS. Among these DVFS techniques, *location_dvfs* still demonstrate the best trade-off point. The 95%-DVFS result improved performance over *location_dvfs* by less than 1%, but gave up 25% in processor lifetime. The 85%-DVFS increased reliability significantly, but more than doubled the performance cost compared to *location_dvfs*.

Our techniques are easily adapted to other sources of asymmetry, such as process variations, as long as we can quantify the effects of such variations on the thermal and power properties of each core.

6.1.3 Hybrid Techniques

We examine the hybrid techniques in this section, and show the results in Figure 7. When we compare the hybrid policies against the DVFS based policies, we see that DVFS-based policies are improved little by combining them with job allocation policies. Again, this is due to the limited gains from reorganizing running threads on a fully utilized system.

6.2 Impact of Partial Utilization

It is expected that most multicore systems will be utilized less than 100% most of the time. This is true especially for the CMPs in the server domain. To evaluate the impact of scheduling mechanisms on reliability when some cores are idle, we used the 12 and 14 thread workloads described in Table 5: a CPU-bound and a mixed CPU-bound/memory-bound workload for each of the 12 and 14 thread cases. The results represent the average of the CPU-bound and mixed cases for the 12 and 14 thread experiments. At the beginning of each simulation, we decided which cores to leave idle

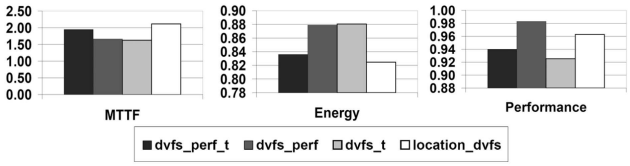

Figure 6: Comparison of DVFS-Based Techniques

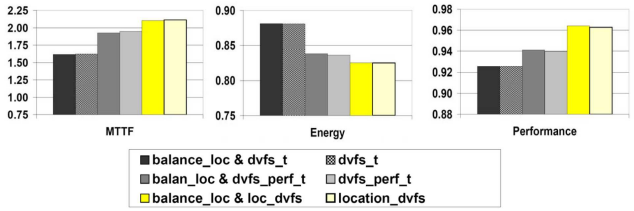

Figure 7: Comparison of Hybrid Techniques

by choosing the allocation with the lowest peak temperature. Once we determined the active cores, we performed the initial placement of threads on these cores randomly.

We first focus on the case with 14 active threads in Figure 8. Although this utilization is close to the full utilization examples explored in Section 6.1, the impact on the reliability of the various policies changes significantly.

Policies with frequent workload re-allocation (i.e., *Balance*, *Migration*) result in poorer reliability with respect to the other policies. The *Balance* policy assigns jobs to cores based on temperature rather than location and often mistakes a core that is cool now for a core that will stay cool in the future. Migration policies that focus heavily on current temperatures are prone to this type of error. The *Migration* result has the same issue. Policies that migrate more than necessary have two distinct reliability disadvantages over the other techniques. First, migrating too often will tend to thwart the DPM manager, which does not put a core to sleep until it has been idle for awhile. This increases the time cores spend running at hotter temperatures. Second, migration causes thermal cycling. This was the dominant cause of the low MTTF results, as the power variations between idle and active states create cycles of a significant magnitude. We examine the effects of migrations in detail in Section 6.4.

The *Stop_Go* policy achieves 1.25 times improvement in MTTF. However, this comes at the cost of a drastic performance and energy cost. Although *Stop_Go* could be utilized effectively as a back-up policy for thermal emergencies to guarantee that temperature does not exceed a given peak value, it is inefficient if used frequently.

Among the DVFS policies, *dvfs_perf* achieves the best performance of less than 2% degradation, while *location_dvfs* results in the longest system life time with a 69% improvement. The hybrid policy *Balance_Location&location_dvfs* seems to provide the best trade-off point among the policies as it achieves almost the same MTTF as *location_dvfs* with better performance and lower energy consumption. The reason the hybrid scheduling policies still provide only small gains over DVFS policies alone is that we start the experiments with an optimal placement of idle cores. We examine this further in Section 6.3.

We expect that as technology scaling continues, the bandwidth for performing voltage scaling will decrease due to the leakage power and transistor threshold voltage limitations. This situation will require other mechanisms for managing power and temperature. *Balance_Location* is our best candidate for workload alloca-

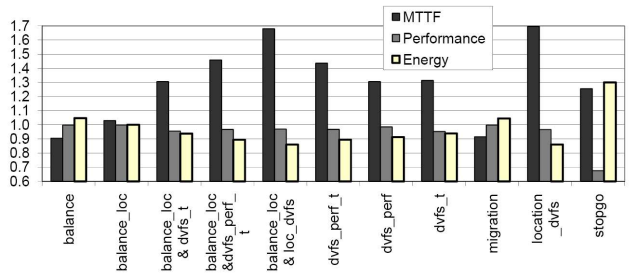

Figure 8: Effect of System Utilization (2 Idle Cores)

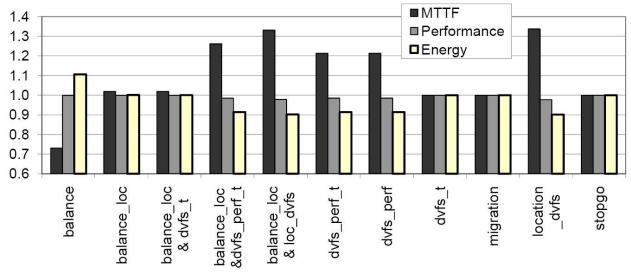

Figure 9: Effect of System Utilization (4 Idle Cores)

tion, as it significantly increases reliability with negligible performance loss.

Figure 9 shows the 12-core utilization results. Because chip temperatures are lower overall, the magnitude of potential reliability gains is reduced. In fact, policies that only react to thermal thresholds see no activity in this scenario (e.g., *Migration*, *dvfs_t*, etc.) and they give the same results as the default policy, as the core temperatures do not exceed the threshold. Policies that proactively look for opportunities can still improve processor lifetime significantly, and even *Balance_Location* provides small gains. Policies that proactively migrate based on current temperature (*Balance*) make mistakes and create thermal cycling.

6.3 Effect of Initial Idle Core Locations

In this section, we examine each policy's ability to adapt to different initial workload mappings on the processor topology. For example, what happens when the initial mapping of threads to cores is highly suboptimal? This could happen with a topology-ignorant scheduler (a likely scenario early on), or just because of jobs entering or leaving the system. We examined several ways of performing the initial allocation: best possible, worst possible, and an in-order placement of jobs on cores.

The *best* case, i.e., the case with the lowest peak temperature, for 12 active threads is leaving the center cores (5, 6, 9, 10) idle, and for 14 active threads when cores 6 and 9 are idle. The *worst* case occurs when the corner cores are idle. Specifically, the worst case for a system with 12 active cores is leaving the cores 0, 3, 12, and 15 idle. Similarly, when 14 cores are active, leaving two of the corner cores on the opposite sides idle, such as cores 0 and 15, represent the worst assignment. The *in-order* initial assignment allocates all available threads on the cores starting from core 0 ascending. This method initially leaves cores 12–15 idle when 12 threads are active, and cores 14–15 idle with 14 threads are active. The *in-order* method attempts to model a naive scheduler that assigns jobs to cores using a *first-available* strategy.

We have observed notable differences in reliability between the experiments. For example, the policy *dvfs_perf_t* experiences a 15% reduction in MTTF in comparison to the best allocation when either the worst or in-order idle core locations are used. This decrease in reliability is comparable to the default policy's 20% reduction in MTTF when using the in-order and worst case initial assignments.

On the other hand, when *dvfs_perf_t* was combined with *Balance_Location*, we were able to achieve a level of reliability to match that of the optimal initial placement. This indicates that one of the major roles of the allocation policy is reassigning thread topologies to assist other policies that optimally set core voltage and frequency. Thus, in a real CMP system, it is critical to combine a *conservative* migration technique (i.e., one which avoids unnecessary migrations and does not create cycling) with DVFS techniques. In the absence of an intelligent migration and scheduling policy, it is difficult to avoid detrimental configurations over time.

6.4 Interactions with Power Management

Dynamic power management (DPM) takes advantage of prolonged core idleness to put the core into a sleep mode. In sleep, the power consumption of the core is greatly diminished. Each of the policies presented is compatible with dynamic power management, but some are able to use DPM opportunities better. Taking a closer look at two extremes, we first examine two policies, *Migration* and *Balance_Location&location_dvfs* for the het_12_mix workload with 12 CPU and memory bound threads. Comparing the thermal traces for *Migration* and *Balance_Location&location_dvfs* (Figure 10), the *Migration* policy suffers significant thermal cycle variations. For *Migration*, we demonstrate the thermal cycles observed on two cores due to frequent re-allocation of workloads. For the *Balance_Location&location_dvfs* policy, we show all the cores' thermal traces, and observe that each core's temperature is stable and lower than the threshold.

This stability along with a lower peak temperature results in significantly higher reliability. *Balance_Location&location_dvfs* turns out to be the best policy when paired with DPM, and it provides an increase in MTTF of 36% over *Migration*, while the performance difference is only 1.5%. So we see that scheduling policies which effectively manage thread locations and DPM policies can reduce processor temperatures and improve reliability. At the same time, DPM can also lead to greater thermal cycling which can counteract some of the MTTF gains that result from the lower power levels of sleeping cores. The adverse effect of DPM on reliability due to thermal cycles is also emphasized in previous work [25]. Thus, when we include the effects of thermal cycling failures, we observe that the traditional assumptions for finding optimal strategies are incomplete; it would be wise to re-design the DPM policies with a reliability perspective.

Despite DPM's possible impact on reliability, we do see (Figure 11) that even in the face of this cycling phenomena, DPM was an overall win for all policies with the exception of *Balance*. In this figure, we show the average results over heterogeneous CPU-bound workloads. The reason that *Balance* received no benefit from using DPM is its proactive mechanism that keeps moving hot threads to colder cores. The result is that no core is idle long enough to trigger the sleep mode. On the other end of the spectrum, *Migration* and *Balance_Location* show gains of 27% and 20% in MTTF for the average case respectively. The reliability improvement in DVFS-based techniques are less prominent and range between 4%–8% MTTF increase.

The policy *Stop_Go* receives a large benefit in energy from using DPM mechanisms, reducing power consumption by 27% in comparison to the no-DPM case. If confronted with a design choice that

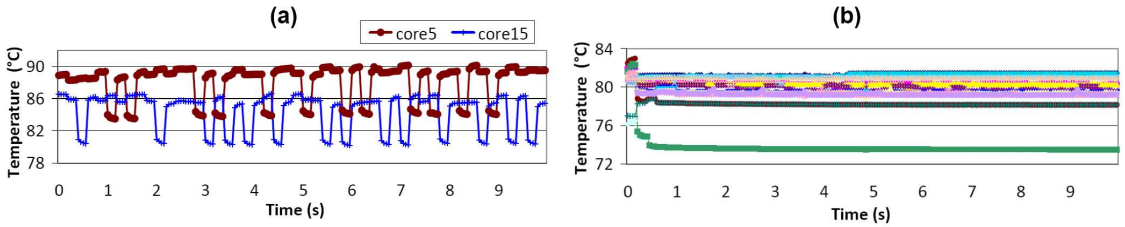

Figure 10: (a) Cycles Caused by the *Migration* Policy; (b) Stable Thermal Profile of *Balance_Location & location_dvfs*

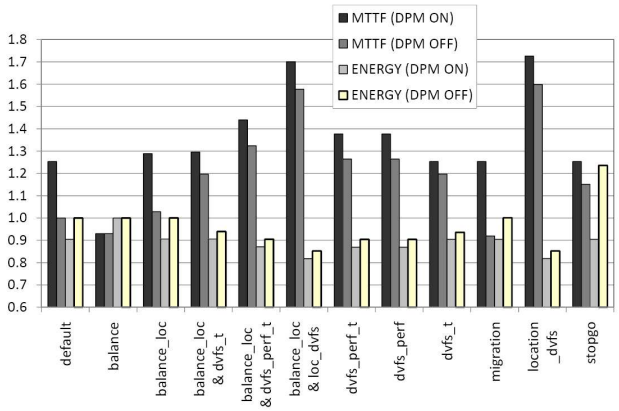

Figure 11: MTTF and Energy Effects of DPM

	Migrations			
	ALL	corner	center	side
balance	4.76	4.75	5.00	4.65
migration	7.66	8.36	5.00	8.66
balance_loc	2.73	1.25	1.60	4.03
balance_loc& dvfs_t	3.65	3.85	2.10	4.33
balance_loc& dvfs_perf_t	3.64	3.86	2.10	4.30
balance_loc& loc_dvfs	3.54	3.70	2.20	4.12
	V/f Setting Changes			
dvfs_perf_t	2.98	2.60	0.80	4.20
dvfs_perf	0.83	1.40	0.00	1.00
dvfs_t	3.40	3.60	0.40	4.80
balance_loc& dvfs_t	3.64	4.40	1.10	4.53
balance_loc& dvfs_perf_t	3.58	4.00	2.00	4.20

Table 6: Number of Migrations and V/f Changes (per Second)

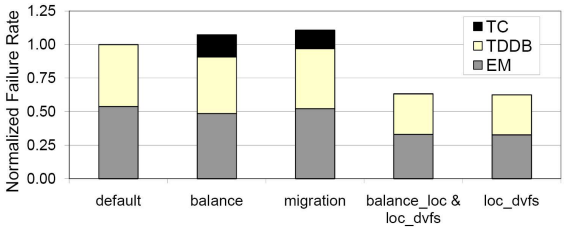

Figure 12: Contributions of Failure Mechanisms

requires the simplicity of *Stop_Go*, DPM could help regain much of the energy lost from the constant start and stop of individual cores.

In Table 6 we show the number of migrations and number of V/f setting changes per second for the policies to provide a more complete understanding of the runtime behavior. The policies that are not listed do not utilize migrations or DVFS. The columns marked as *ALL, corner, center*, and *side* refer to the average number across all cores, across only the corner cores, center cores and side cores, respectively. The results are with DPM, and for the CPU-bound heterogeneous workload with 14 threads (i.e., 2 idle cores). *Migration* has a significantly higher number of thread movements in comparison to other policies: almost 3 times more than *Balance_Location*. The low migration count of *Balance_Location* is a result of its ability to match the performance characteristics of applications with the thermal behavior of cores. Compared to *Balance_Location* only, combining *Balance_Location* with DVFS increases the frequency of migrations, as the temperature profile of the cores vary more when their V/f settings are dynamically adjusted. Among the DVFS policies, *dvfs_perf* has the lowest number of changes as it only alters the V/f setting of applications tolerant to operating at a slower speed. Also, *dvfs_perf_t* reduces the frequency of changes in comparison to *dvfs_t* as it proactively adjusts the V/f setting and triggers the thermal threshold fewer times.

To better understand the tension between the different failure mechanisms, Figure 12 presents a breakdown of the contribution of different failure types to reliability. This figure demonstrates the normalized average failure rate for our two best and two worst policies (i.e., best/worst in terms of their average MTTF results). The workload for this experiment is the heterogeneous CPU-bound workload with 12 threads. Recall that the failure rate is inversely proportional to MTTF. This figure shows that *Balance* and *Migration* reduce the probability of failures due to electromigration (EM) and dielectric breakdown (TDDB). If we ignored the effect of thermal cycles, we would conclude that reliability had increased. However, because of the number of thread migrations, they create large thermal cycles (TC). The *Location_dvfs* and the hybrid *Balance_Location&Location_dvfs* policies, on the other hand, reduce the failure rates caused by thermal hot spots without introducing a significant amount of thermal cycling failures. Note that in the default case, as there is no workload re-allocation, temperature is stable and no cycles are observed.

7. CONCLUSIONS

This paper analyzes how job scheduling and power management policies affect system lifetime. It demonstrates a novel CMP simulation framework which is able to simulate thermal dynamics over far longer time periods than typical architectural simulators, at high accuracy. It evaluates a number of techniques in terms of their effect on reliability, temperature, energy, and performance.

The results in this paper provide several key insights that will serve us well in the design of future thermal management policies:

- It is critical to consider thermal cycling effects in addition to peak temperature effects. We saw two policies that erroneously appear to increase lifetime when thermal cycling was ignored.
- Thermal cycling is not a significant effect in a fully utilized system, as the variance in power between running threads was not

shown to be sufficiently high to cause harmful effects. However, when cores are idle, it is important that we manage the idle cores in a way that does not exacerbate thermal cycling.

- Conservative policies that minimize migration not only reduce thermal cycling, but also maximize our ability to exploit sleep states via DPM.
- Understanding thermal asymmetries, which are either due to the layout of the processor or due to process variation, is critical to effective thermal management. Not understanding thermal variance causes much unnecessary movement because we cannot discern between a hot thread and a hot core. Understanding the thermal variance allows us to employ an asymmetric thermal policy that accounts for and even exploits that asymmetry.
- Proactive techniques that apply DVFS to frequency-tolerant applications can raise the performance of the entire system. This is somewhat non-intuitive, as the frequency-tolerant applications are also the coolest applications. However, by lowering overall temperatures chip-wide, this allows the hot applications to run longer without triggering thermal events.

In future work, we will be addressing reliability management of multithreaded multicore systems. We will seek to provide a comprehensive understanding of how parallel workloads differ from single-threaded benchmarks, and propose novel management techniques to address the particular characteristics of such workloads.

Acknowledgment

The authors would like to thank Erez Zadok as well as all of the anonymous reviewers for their help in improving the paper. This research was supported in part by NSF grant CCF-0702349, GSRC, NSF GreenLight, Cisco, CNS, Sun Microsystems, and UC Micro grant 06-198.

8. REFERENCES

[1] M. V. Biesbrouck, T. Sherwood, and B. Calder. A co-phase matrix to guide simultaneous multithreading simulation. In *International Symposium on Performance Analysis of Systems and Software*, pages 45–56, 2004.

[2] N. L. Binkert, E. G. Hallnor, and S. K. Reinhardt. Network-oriented full-system simulation using M5. In *Computer Architecture Evaluation using Commercial Workloads (CAECW)*, 2003.

[3] P. Bose. Power-efficient microarchitectural choices at the early design stage. In *Keynote Address, Workshop on Power-Aware Computer Systems*, 2003.

[4] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *High-Performance Computer Architecture (HPCA)*, pages 171–182, 2001.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *International Symposium on Computer architecture*, pages 83–94, 2000.

[6] D. C. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report CS-TR-97-1342, University of Wisconsin, Madison, June 1997.

[7] J. Chang and G. S. Sohi. Cooperative cache partitioning for chip multiprocessors. In *ACM International Conference on Supercomputing*, pages 242–252, 2007.

[8] A. K. Coskun, T. Rosing, and K. Whisnant. Temperature aware task scheduling in MPSoCs. In *Design Autom. and Test in Europe (DATE)*, pages 1659–1664, 2007.

[9] G. Dhiman and T. Rosing. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In *International Symposium on Low Power Electronic Design (ISLPED)*, pages 207–212, 2007.

[10] J. Donald and M. Martonosi. Leveraging simultaneous multithreading for adaptive thermal control. In *Second Workshop on Temperature-Aware Computer Systems*, 2005.

[11] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *International Symposium on Computer Architecture (ISCA)*, pages 78–88, 2006.

[12] M. Gomaa, M. D. Powell, and T. N. Vijaykumar. Heat-and-Run: leveraging SMT and CMP to manage power density through the operating system. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 260–270, 2004.

[13] S. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the impact of increasing microprocessor power consumption. In *Intel Technology Journal*, 2001.

[14] S. Heo, K. Barr, and K. Asanovic. Reducing power density through activity migration. In *ISLPED*, pages 217–222, 2003.

[15] Intel pentium 4 processor in the 423-pin package thermal design guidelines. Technical Report 249203-001, Intel, November 2000.

[16] R. Iyer et al. QoS policies and architecture for cache/memory in CMP platforms. In *ACM Sigmetrics*, pages 25–36, 2007.

[17] Failure mechanisms and models for semiconductor devices, JEDEC publication JEP122C. http://www.jedec.org.

[18] A. Karlin, M. Manesse, L. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. In *Algorithmica*, pages 542–571, 1994.

[19] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha. HybDTM: a coordinated hardware-software approach for dynamic thermal management. In *Design Automation Conference (DAC)*, pages 548–553, 2006.

[20] R. Kumar, D. M. Tullsen, and N. P. Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *Parallel Architectures and Compilation Techniques (PACT)*, pages 23–32, 2006.

[21] R. Kumar, V. Zyuban, and D. M. Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *ISCA*, pages 408–419, 2005.

[22] S. Murali et al. Temperature control of high-performance multicore platforms using convex optimization. In *Design Autom. and Test in Europe (DATE)*, pages 110–115, 2008.

[23] H. Nguyen. Multilevel interconnect reliability on the effects of electro-thermomechanical stresses. Ph.D. dissertation, University of Twente, Netherlands, 2004.

[24] M. Powell, E. Schuchman, and T. Vijaykumar. Balancing resource utilization to mitigate power density in processor pipelines. In *International Symposium on Microarchitecture*, pages 294–304, 2005.

[25] T. S. Rosing, K. Mihic, and G. D. Micheli. Power and reliability management of SoCs. In *IEEE Transactions on VLSI*, 15(4), pages 391–403, April 2007.

[26] T. Sherwood, G. H. E. Perelman, and B. Calder. Automatically characterizing large scale program behavior. In *ASPLOS*, 2002.

[27] K. Skadron. Hybrid architectural dynamic thermal management. In *Design Autom. and Test in Europe (DATE)*, pages 10–15, 2004.

[28] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *ISCA*, pages 2–13, 2003.

[29] J. Smith. Characterizing computer performance with a single number. In *Communication of ACM, 31(10)*, pages 1202–1206, 1988.

[30] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The case for lifetime reliability-aware microprocessors. In *ISCA*, pages 276–287, 2004.

[31] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The impact of technology scaling on lifetime reliability. In *International Conference on Dependable Systems and Networks (DSN)*, pages 177–186, 2004.

[32] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. Lifetime reliability: Toward an architectural solution. In *IEEE Micro*, 25(3):70–80, 2005.

[33] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif. Full-chip leakage estimation considering power supply and temperature variations. In *ISLPED*, pages 78–83, 2003.

[34] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. CACTI 4.0. Technical Report HPL-2006-86, HP Laboratories Palo Alto, 2006.

[35] Q. Wu, M. Martonosi, D. W. Clark, V. J. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks. A dynamic compilation framework for controlling microprocessor energy and performance. In *International Symposium on Microarchitecture*, pages 271–282, 2005.