Static and Dynamic Temperature-Aware Scheduling for Multiprocessor SoCs

Ayşe Kıvılcım Coşkun, *Student Member, IEEE*, Tajana Šimunić Rosing, *Member, IEEE*, Keith A. Whisnant, *Member, IEEE*, and Kenny C. Gross, *Member, IEEE*

***Abstract*—Thermal hot spots and high temperature gradients degrade reliability and performance, and increase cooling costs and leakage power. In this paper, we explore the benefits of temperature-aware task scheduling for multiprocessor system-on-a-chip (MPSoC). We evaluate our techniques using workload characteristics collected from a real system by Sun's Continuous System Telemetry. We first solve the task scheduling problem statically using integer linear programming (ILP). The ILP solution is guaranteed to be optimal for the given assumptions for tasks. We formulate ILPs for minimizing energy, balancing energy, and reducing hot spots, and provide an extensive comparison of their thermal behavior against our technique. Our static solution can reduce the frequency of hot spots by 35%, spatial gradients by 85%, and thermal cycles by 61% in comparison to the ILP for minimizing energy. We then design dynamic scheduling policies at the OS-level with negligible performance overhead. Our adaptive dynamic policy reduces the frequency of high-magnitude thermal cycles and spatial gradients by around 50% and 90%, respectively, in comparison to state-of-the-art schedulers. Reactive thermal management strategies, such as thread migration, can be combined with our scheduling policy to further reduce hot spots, temperature variations, and the associated performance cost.**

***Index Terms*—Continuous system telemetry, multiprocessor scheduling, reliability, thermal management.**

I. Introduction

ADVANCES in process technology enable manufacturing multiprocessor SoCs including CPUs, memories and communication architectures on a single die. The Sun UltraSPARC T1 [15] and the IBM Cell [13] are examples of such processors. However, in deep submicrometer process technologies, high power and temperature densities, process imperfections, and reduced voltage margins have made the systems much more vulnerable to failures. In addition to degrading reliability, thermal hot spots and high temperature gradients bring challenges for performance, cooling costs, and leakage power. In this paper, we propose static and dynamic temperature-aware MPSoC scheduling strategies. We show that our static solution, which targets both hot spots and gradients, results in significantly better thermal profiles in comparison to other energy or thermal-based static methods. For dynamic scheduling in MPSoCs, we introduce an adaptive technique, which can achieve low and stable temperature profiles without noticeable impact on performance.

Thermal hot spots increase cooling costs, negatively impact reliability and degrade performance. The significant increase in cooling costs requires designing for temperature margins that are lower than the worst-case. Hot spots accelerate failure mechanisms such as electromigration, stress migration, and dielectric breakdown, which cause permanent device failures [12]. In fact, a small difference in the operating temperature (i.e., 10 °C–15 °C) can result in a 2× difference in the mean time to failure of the devices [40]. Leakage is exponentially related to temperature, and a positive feedback loop exists between temperature and leakage, which can cause dramatic increases in temperature and damage the circuit if not controlled. High temperatures also adversely affect performance, as the effective operating speed of devices decreases with high temperature.

Addressing thermal hot spots alone is not enough to achieve better reliability, and temperature gradients in time and space determine device reliability at moderate temperatures [19]. The failure rate due to thermal cycling increases with the increasing magnitude and frequency of the temperature cycles [12]. A 10 °C increase in the magnitude of cycles can cause about 16× decrease in mean time to failure for metallic structures [12]. Thermal cycling causes accelerated package fatigue and plastic deformations of materials, and leads to permanent failures. Such cycles are created either by low frequency power changes such as system power on/off cycles, or by workload rate changes and power management decisions, which happen much more frequently [27].

Large spatial temperature variations across the chip cause performance or logic failures. Negative bias temperature instability (NBTI) and hot carrier injection (HCI) cause the circuits to fail in meeting timing constraints [16]. In process technologies below 0.13 μm, reliability issues arise due to NBTI and HCI, as the operating temperatures and electric fields reach high enough values to accelerate these mechanisms during device lifetime. In addition, increasing temperature increases local resistances, and thus circuit delays and IR drop [30]. Global clock networks are especially vulnerable to spatial variations. Every 20 degrees increase in temperature causes 5%–6% increase in Elmore delay in interconnects. As a result, clock skew problems become noticeable for spatial variations of even 15–20 degrees [1].

To date, temperature related problems have been addressed to a limited extent by techniques that lower the average temperature or keep the temperature under a given threshold. Power-aware synthesis, dynamic power management (DPM), and dynamic voltage scaling (DVS) and dynamic thermal management

Manuscript received May 31, 2007; revised August 15, 2007. First published August 4, 2008; last published August 20, 2008 (projected). This work was supported in part by Sun Microsystems and by the University of California MICRO under Grant 06-198.

A. K. Coşkun and T. S. Rosing are with the Computer Science and Engineering Department, University of California San Diego, La Jolla, CA 92093 USA (e-mail: acoskun@ucsd.edu; trosing@ucsd.edu; tajana@ucsd.edu).

K. A. Whisnant and K. C. Gross are with Sun Microsystems, San Diego, CA 92121 USA (e-mail: kwhisnan@myrealbox.com; kenny.gross@sun.com).

Digital Object Identifier 10.1109/TVLSI.2008.2000726

are such techniques. A significant bottleneck of such methods is the performance impact associated with stalling or slowing down the processor [33]. When the workload that is going to run on the system is known (as in some embedded systems), voltage/frequency levels or architecture configuration can be adjusted at the design stage to avoid dynamic thermal management as much as possible [35]. Despite their significant benefits to the thermal profile of the chip, conventional power or thermal management techniques cannot always eliminate the problems associated with temperature in a cost-effective way. Both temporal and spatial temperature variations cause a number of reliability problems, while state-of-the-art power and thermal management techniques do not focus on the effects of these variations.

In this paper, we first propose an integer linear programming (ILP) based static scheduling method that minimizes both thermal hot spots and temperature gradients to increase MPSoC reliability. We formulate ILPs for minimizing energy, balancing energy and minimizing hot spots (without considering gradients), and provide an extensive comparison of their thermal behavior against our technique. Our solution reduces the frequency of hot spots by 35%, spatial gradients by 85%, and thermal cycles by 61% in comparison to the ILP for minimizing energy. The static approach sets a baseline for comparison of dynamic scheduling techniques. It can also be utilized for systems where workload can be estimated accurately *a priori*.

We then develop dynamic OS-level scheduling policies. In contrast to thermal management techniques, which perform computation migration or clock gating (e.g., [32]) when temperatures reach critical values, our goal is to adjust the workload distribution to achieve better temporal and spatial thermal profiles. We evaluate heuristics that make decisions based on the current temperature, and we propose a probabilistic scheduling technique, *Adaptive-Random*. This technique adapts to the temperature changes by taking the thermal history into account. Our analysis shows that Adaptive-Random can reduce the temporal and spatial gradients by 50% and 90%, respectively, in comparison to state-of-the-art schedulers. When combined with reactive methods such as thread migration or voltage scaling, the Adaptive-Random policy decreases the performance impact of such techniques to below 7%, while achieving better thermal profiles. In our experimental evaluation, we leverage Sun's patented Continuous System Telemetry Harness (CSTH) [9] to collect and analyze realistic workload information on a real system.

The rest of this paper starts with a discussion of the related work. We then explain our static and dynamic temperature-aware scheduling approaches in Section III and in Section IV, respectively. The experimental methodology and results are provided in Sections V and VI. We conclude in Section VII.

II. Related Work

A. Scheduling

In the literature, a number of power-aware scheduling techniques have focused on voltage scheduling in single-processor systems. In [6], a DVS method based on decomposing on-chip and off-chip workload is presented. Kim *et al.* compare the performance of well-known DVS algorithms in [14]. Quan *et al.* introduce fixed priority DVS scheduling techniques with constant and variable voltage levels [25].

In multicore systems, performing power-aware scheduling while ensuring timing and performance constraints introduce high complexity. A power management strategy for mission critical systems containing heterogeneous devices is proposed in [21]. A static scheduling method optimizing concurrent communication and task scheduling for heterogeneous network-on-chips is proposed in [10]. Rong *et al.* formulates the ILP for finding the optimal voltage schedule and task ordering for a system with a single core and peripheral devices [26]. In [28], MPSoC scheduling problem is solved with the objectives of minimizing the data transfer on the bus and guaranteeing deadlines for the average case, using ILP and constraint programming, respectively. Minimizing energy on MPSoCs using DVS has been formulated using a two-phase framework in [42]. In [24], a functional unit rotation approach and load balancing are combined with low power scheduling to reduce temperature in VLIW processors.

In this paper, we propose temperature-aware task scheduling strategies for MPSoCs. First, using integer linear programming (ILP), we formulate the problem of finding an optimal task allocation on an MPSoC for minimizing and balancing temperature, while maintaining performance and precedence constraints of jobs. Our paper differs from [26], as we look into systems with multiple processing units and optimize the system for not only power but also for temperature. Our thermally-aware scheduling approach achieves the best possible temperature profile for a given performance constraint.

For MPSoCs in high end computing domain, where workload is not known *a priori*, making dynamic and fast decisions for workload allocation is crucial for performance. Thus, the dynamic techniques we focus on are low overhead OS level scheduling methods, as opposed to high complexity strategies with aggressive performance and energy optimization goals. Next, we discuss the MPSoC thermal management strategies, and compare our dynamic scheduling techniques to conventional dynamic thermal management.

B. Thermal Modeling and Management

As power-aware policies are not always sufficient to prevent temperature induced problems, thermal modeling and management methods have been proposed. HotSpot [33] is an automated thermal model, which calculates transient temperature response given the physical characteristics and power consumption of units on the die. A fast thermal emulation framework for field-programmable gate arrays (FPGAs) is introduced in [2], which reduces the simulation time considerably while maintaining accuracy. Static methods for thermal and reliability management are based on thermal characterization at design time. In [35], hot spots are predicted based on the execution profile of the multimedia benchmarks. A simulated-annealing-based thermal floorplanning method at microarchitecture level is presented in [29]. Including temperature as a constraint in the co-synthesis framework and in task allocation for platform-based system design is introduced

TABLE I
SUMMARY OF ALL THE ILP OBJECTIVE FUNCTIONS

Label	ILP Objective	Objective Equation
Min-Th&Sp	Minimizing thermal hot spots and gradients	Minimize $H+G$; $H = max\{Q_p; p = 1...m,$ for a system of m cores$\}$ where: $Q_p = \sum_{T_i \in T} \{x_{ip} \sum_{v_k} (q_{ik} y_{ik})\}$ $G = \sum_{p,r \in PU, p \neq r} \{n_{pr}\{ \sum_{i,j \in T, i \neq j} x_{ip} x_{jr} [p_{ij} d_{ij} (\tau_i - s_j) + p_{ji} d_{ji} (\tau_j - s_i)]\}\}$
Min-Th	Minimizing & balancing thermal hot spots	Minimize H; $H = max\{Q_p; p = 1...m,$ for a system of m cores$\}$ where: $Q_p = \sum_{T_i \in T} \{x_{ip} \sum_{v_k} (q_{ik} y_{ik})\}$
Bal-En	Balancing energy consumption	Minimize EN_{max}; $EN_{max} = max\{EN_p; p = 1...m,$ for a system of m cores$\}$ where: $EN_p = \sum_{T_i \in T} \{x_{ip} \sum_{v_k} (e_{ik} y_{ik})\}$
Min-En	Minimizing total energy	Minimize EN_{total}; $EN_{total} = \{\sum_{T_i \in T} \sum_{v_k} e_{ik} y_{ik}\} + I_{total}$; $I_{total} = \sum_{p \in PU} \{ \sum_{i,j \in T, i \neq j} x_{ip} x_{jp} m_{ij} \; idle(s_j - \tau_i)\}$

in [11]. Reliability aware microprocessor (RAMP) provides a reliability model at architecture level for temperature related intrinsic hard failures [36]. It analyzes the effects of application behavior on reliability and optimizes the architectural configuration and power/thermal management policies for reliable design. In [27], it is shown that aggressive power management can adversely affect reliability due to fast thermal cycles, and the authors propose an optimization method for MPSoCs that can save power while meeting the reliability criteria.

Dynamic thermal management controls overheating by keeping the temperature below a critical threshold. Computation migration and fetch toggling are examples of such techniques [33]. Heat-and-Run performs temperature-aware thread assignment and migration for multicore multithreaded systems [8]. Kumar *et al.* propose a hybrid method that coordinates clock gating and software thermal management techniques [17]. The multicore thermal management method introduced in [7] combines distributed DVS with process migration. In [37], dynamic MPSoC temperature-aware scheduling methods that take core temperatures into account while making decisions are proposed.

The dynamic scheduling methods we propose optimize the thermal profile of the MPSoC with minimal impact on performance. By making decisions based on the temperature readings, we show that hot spots and high magnitudes of temporal and spatial variations can be mostly prevented. Our technique can be combined with more aggressive thermal management strategies to reduce their performance cost while further lowering and balancing the temperature. Our work in dynamic temperature-aware scheduling differs from previous thermal management techniques, as we propose a policy that adapts to changes by probabilistically assigning workload to cores based on their temperature history. We next describe our static and dynamic scheduling techniques in Sections III and IV.

III. OPTIMAL STATIC SCHEDULING

In this section, we describe a static task scheduling approach for minimizing the frequency of thermal hot spots and large temperature gradients in order to increase system reliability and ease the temperature related design challenges. In addition to our technique, we formulate static scheduling for minimizing energy, balancing energy, and minimizing the thermal hot spots (i.e., without considering temperature gradients). This way, for the first time, we provide a comparison of various static scheduling techniques in terms of their efficiency in handling thermal issues.

Our goal in static optimization is finding a task schedule for the MPSoC where the deadline and dependence constraints of tasks are met, and the best possible temperature profile is achieved throughout the execution. Achieving the best temperature profile corresponds to minimizing and balancing the temperature across the MPSoC as discussed in Section I. Addressing both the hot spots and gradients is our solution's (demonstrated as `Min-Th&Sp`) distinguishing feature from the other energy and thermal-based static strategies shown in Table I. Our method utilizes ILP, which is commonly used for solving scheduling problems (e.g., [28]). The ILP solution gives optimal results assuming the estimates for task execution times, deadline, and temperature profile are accurate. We provide the objective functions of all the ILPs we solve in this paper in Table I. Minimizing and balancing the hot spots (`Min-Th`) reduces the thermal hot spots, but there is no consideration for spatial gradients. For energy balancing (`Bal-En`), the ILP minimizes the maximum energy consumption for each core, which balances the energy profile on the system. The ILP for minimizing energy (`Min-En`) minimizes the cumulative sum of all the active and idle state energy. Next, we explain the formulations in detail.

In our system model, we assume the MPSoC contains m processor units, $PU = PU_p$; $p = 1, \ldots, m$, and we model the applications using task graphs. In the graph $G = (T, E)$, each vertex represents a task $(T_i \in T)$, which is a function or collection of functions to be performed. Each edge in the graph $(E_{ij} \in E)$ shows that task T_j is dependent on T_i in order to perform computation. A simple example task graph is shown in Fig. 1. We assume the deadlines (D_i) and worst-case execution times $(WCET_i)$ of tasks are known *a priori*.

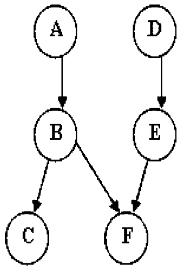

Fig. 1. Example task graph.

TABLE II
VARIABLES USED IN THE ILP

x_{ip}:	Set of 1-0 variables s.t. $x_{ip} = 1$ iff T_i is assigned to PU_p
q_{ik}:	Time spent above threshold temperature while running T_i at v_k
t_i:	WCET of T_i considering the voltage setting
s_i:	Execution start time for T_i
τ_i:	Execution finish time for T_i
p_{ij}:	Set of 1-0 variables s.t. $p_{ij} = 1$ iff T_i starts before T_j
n_{pr}:	Set of 1-0 variables s.t. $n_{pr} = 1$ iff p and r are adjacent cores
d_{ij}:	Set of 1-0 variables s.t. $d_{ij} = 1$ iff $\tau_i \geq s_j$
y_{ik}:	Set of 1-0 variables s.t. $y_{ik} = 1$ iff T_i runs at speed v_k
m_{ij}:	Set of 1-0 variables s.t. $m_{ij} = 1$ iff T_j immediately follows T_i
	* *s.t.: such that*

TABLE III
ILP FORMULATION FOR Min-Th&Sp

Minimize $H + G$; $H = max\{Q_p; p = 1...m$, for a system of m cores$\}$ where: $Q_p = \sum_{T_i \in T} \{x_{ip} \sum_{v_k} (y_{ik} q_{ik})\}$ $G = \sum_{p,r \in PU, p \neq r} \{n_{pr}\{\sum_{i,j \in T, i \neq j} x_{ip} x_{jr} [p_{ij} d_{ij} (\tau_i - s_j) + p_{ji} d_{ji} (\tau_j - s_i)]\}\}$ Subject to constraints:	
(a) $\forall T_i : \sum_p x_{ip} = 1$	Each task is assigned to only one PU
(b) $\forall T_i : \sum_k y_{ik} = 1$	Each task runs at only one V/f level
(c) $\tau_i = s_i + t_i$	Execution finish time for T_i
(d) $s_i \geq max_{E_{ji} \in E}\{\tau_j\}$	Task precedence
(e) $\tau_i \leq D_i$	Deadlines for all sink nodes
(f) $s_i \geq \tau_j;\ if\ p_{ji} = 1$	Precedence for tasks on the same core
(g) $p_{ij} + p_{ji} = 1$; if $x_{ip} = x_{jp} = 1$	If T_i and T_j are scheduled on the same core, either T_i precedes T_j, or vice versa

We assume each PU has v discrete voltage settings $V = V_k$; $k = 1, \ldots, v$ (in decreasing order). Each voltage setting (V_k) can be associated with a computation speed v_k in terms of cycles/second. Thus, the energy consumption for executing T_i at speed v_k can be expressed as $e_{ik} = (g_{\text{active}}(v_k) * t_i)$, where g_{active} is the function for power consumption and t_i is the execution time of T_i. The power consumed during task execution is a monotonically increasing and convex function of the computation speed [41]. Assuming the tasks execute up to their WCET and WCET_i is given as the execution time in the default (highest) processor frequency, the WCET for T_i running at processor speed v_k is computed as: $t_i = \text{WCET}_i * (v_1/v_k)$.

The objective function of our ILP (Min-Th&Sp) has two parts: 1) minimizing and balancing the thermal hot spots (H in Table I) and 2) minimizing the spatial gradients (G). The first part of the objective function minimizes the maximum time spent above the threshold for each core, which balances the thermal hot spots across the chip. Consequently, it reduces the magnitude of temporal variations in temperature. However, it does not consider the spatial gradients on the die.

Spatial gradients increase when several jobs are clustered in neighboring units, while the rest of the units are idle. Contrarily, when the workload is spatially spread out on the die, because of the effect of heat transfer from hot to cool cores, more even temperature distributions are achieved. For example, a checkerboard arrangement of the workload where each active core has an idle neighbor reduces the spatial gradients in comparison to having all active cores on one half of the MPSoC and idles ones on the other half. Therefore, reducing the high spatial differentials requires avoiding scheduling jobs in neighboring cores at the same time. We call this situation *overlap*, where two jobs are scheduled in next-door neighbors at the same time. Thus, in the second part of the objective function (G in Table I), we minimize the total overlap. We only take into account side-by-side neighbors in our overlap computations, as it is shown that heat sharing among neighbor units with adjoining sides have a significant effect on temperature [33]. Minimizing the sum of H and G addresses both balancing thermal hot spots and minimizing the spatial gradients.

Table III provides the complete formulation of the ILP for Min-Th&Sp, and the variables used in the formulation are defined in Table II. In the first part of objective function, (H), we minimize the total time spent above a given threshold in order to eliminate the hot spots and balance temperature. We use "time spent over a temperature threshold" (represented with q_{ik}) as a metric for profiling the thermal behavior of tasks, and also as a metric for evaluating our ILP. This metric has also been used in previous work as well [18]. Minimizing the average or peak temperature does not adequately capture the temperature profiles. For example, a system that experiences 90 °C temperature for an hour is expected to have worse reliability than a system that has 90 °C (i.e., same peak temperature) for a second. Or, two systems can have the same average temperature but very different thermal profiles. Having a time-based metric addresses such differences.

To compute the time spent above the threshold for each task (q_{ik}), we perform thermal simulations. The temperature profile for a job depends on the allocation and the floorplan of the MPSoC in addition to the individual power characteristics of the job. We initially assume the time spent above the threshold for each task is equal to the execution time (WCET) of the task ($q_{ik} = \text{WCET}_i$). We solve the ILP (Min-Th&Sp) for the given task graph, maintaining the deadlines and precedence constraints among jobs. We next perform thermal simulation and record the time spent above the threshold temperature for each task. Afterwards, we insert these new q_{ik} estimates in the ILP, and solve the ILP again to get the final scheduling results. We set the threshold temperature to 85 °C in our simulations, as 85 °C is considered a high temperature for many processors [31]. To verify the accuracy of this thermal estimation method, we performed simulations on five randomly generated task sets, each set containing ten tasks. First, we solved the ILP using the thermal estimates we obtained as described above. We then simulated the temperature response for the schedule determined by the ILP, and recorded the time spent above the threshold. Iterating on this loop several times (i.e., getting new q_{ik} estimates from thermal simulation, feeding them into the ILP, and solving the ILP again with the new q_{ik} values), we have observed that the error in temperature estimation stays below 5%.

In the computation of the Q_p component of H, y_{ik} is an integer variable that is 1 *iff* task T_i is scheduled to run at frequency v_k and q_{ik} is the thermal estimate for T_i at frequency v_k. The ILP formulation we provide here for Min-Th&Sp can be applied to systems that have dynamic power management (DPM) or dynamic voltage scaling (DVS). For the cases without power management or DPM only, there is only one voltage setting, so $y_{i1} = 1$. For DPM, the q_{ik} estimates should be derived through simulations with DPM, as putting the cores into sleep state is expected to affect the thermal behavior. For DVS cases, we assume that each task will run on a fixed frequency. We then need to evaluate the thermal profile of each task for all frequency/voltage levels.

In the second part of the objective function of Min-Th&Sp (G in Table III), we compute the total *overlap* in the schedule. We use two additional variables while formulating the overlap: n_{pr} is an integer variable which is equal to 1 only if cores p and r are adjacent to each other in the floorplan; d_{ij} is an integer variable that is equal to 1 only if the completion time of T_i is greater than the start time of T_j. The x variables check if two tasks are scheduled on neighbor units. The product $p_{ij}.d_{ij}$ is equal to 1 if T_i precedes T_j and if T_j starts before T_i finishes. So, when $p_{ij}.d_{ij} = 1$, there is an overlap of T_i and T_j. The difference $t_i - s_j$ quantifies the duration of the overlap.

Table III also demonstrates the ILP constraints that guarantee the deadlines and task precedence. The x integer variables defined in (a) assure that each task is assigned to only one core and (b) shows that each task runs at a fixed voltage setting. Constraint (c) computes the finish time of tasks to guide the precedence and deadline constraints. We use two sets of precedence constraints. The first set, (d), makes sure the dependences between tasks are satisfied, so that a consumer task does not start before all its producer tasks complete. In addition to this, if several tasks are scheduled on the same core, a task can only start after the previously scheduled tasks are completed (f). We define the p variables in (g) to help defining the constraints in (f). Constraint (e) ensures that all tasks with deadlines finish before their deadlines. We next provide the details for the other ILP formulations we presented in Table I, and point out their differences with Min-Th&Sp.

Minimizing and balancing the thermal hot spots: Min-Th minimizes the maximum time spent above threshold temperature for each core in order to minimize and balance the thermal hot spots. In the ILP formulation for Min-Th, the second part of the objective function is eliminated (i.e., G = 0) as this ILP does not consider spatial gradients. The rest of the formulation is the same.

Energy balancing: The ILP formulation for Bal-En has the overlap set to zero ($G = 0$), and the temperature variable q_{ik} in Min-Th&Sp is replaced with e_{ik}, which is the energy per task for running T_i at frequency v_k. y_{ik} is an integer variable that is 1 *iff* T_i runs at frequency v_k. Summing the expression $e_{ik}y_{ik}$ computes the energy consumed per task.

Minimizing energy: The ILP for Min-En is applicable to systems with DPM, DVS, or both. For systems with DPM only, the ILP is solved for only the default frequency of the system. For DVS, the $\sum_{v_k} e_{ik}y_{ik}$ term computes the energy per task for the frequency level task T_i is assigned. In that case, the timing parameters (e.g., t_i, s_i, etc., in Table II) are computed considering the voltage settings of tasks.

While computing the total energy EN_{total}, we consider the energy consumed during all active and idle periods. In order to compute the length of idle time slots, we define an integer variable, m_{ij}, which is 1 iff task T_i starts before T_j, and there is no other task whose start time is between s_i and s_j. Minimizing the total energy involves minimizing the energy consumption during idle time slots as well. I_{total} is the energy spent during all idle times, and the *idle(y)* function computes the energy for individual idle time slots. For example, when we apply a fixed timeout dynamic power management (DPM) strategy which puts cores into sleep state if the idle time is longer than a given timeout ($t_{timeout}$), the energy during idle time slot S can be computed as follows. Here, e_{penalty} and t_{penalty} are the energy and time overhead for switching into and out of the sleep state, respectively

$$idle(S) = e_{\text{penalty}} + e_{slp}(S - t_{\text{penalty}}) \text{ if } S \geq t_{\text{timeout}} \quad (1)$$

$$idle(S) = e_{idle}.S \text{ if } S < t_{\text{timeout}}. \quad (2)$$

We have described the detailed formulations of the ILPs for Min-Th&Sp, Min-Th, Min-En, and Bal-En. These ILP formulations include multiple nonlinear equations. Solutions to similar nonlinear problems have been presented in [26] and [41]. Such problems can be solved by ILP solvers after linearization using standard techniques [5]. We next describe the linearization techniques we used.

First, when two integer variables are multiplied, we introduce a third variable instead of the product, and create additional constraints to define this variable. Assuming variables x_{ip} and x_{ir} are multiplied, we add a third 0–1 variable X_{ipjr} in the formulation with the following constraints:

$$x_{ip} + x_{jr} - X_{ipjr} \leq 1 \quad (3)$$

$$-x_{ip} - x_{jr} + 2X_{ipjr} \leq 0. \quad (4)$$

When multiplying binary (1-0) variables with integer values such as $(p_{ij}.s_i)$, we use the following linearization, where D is a suitably large bound for the variables. We define a new variable r_{ij}, such that $r_{ij} = p_{ij}.s_i$. The following constraints satisfy the multiplication:

$$r_{ij} - D \cdot p_{ij} \leq 0 \quad (5)$$

$$-s_i + r_{ij} \leq 0 \quad (6)$$

$$s_i - r_{ij} + D \cdot p_{ij} \leq D. \quad (7)$$

To linearize the step function introduced by the d_{ij} variables, we use (8). The multiplications in this equation are linearized using the methods previously discussed

$$d_{ij}(\tau_i - s_j) + (1 - d_{ij})(-\tau_i + s_j) \geq 0. \quad (8)$$

Though converting the nonlinear problem to a linear one is possible through these techniques, including integer variables in the formulation causes the problem size to grow exponentially as the number of tasks increase. As a result, it may become infeasible to solve an allocation problem for many tasks. For large task sets, ILPs can be solved using LP relaxation and randomized rounding techniques [39]. For many NP-hard problems, randomized rounding is shown to yield the best approximation known by any polynomial time algorithm [39].

In this section, we presented a methodology for statically optimizing task scheduling with temperature-aware objectives. We also formulated other energy and thermal based ILPs for comparison purposes. In Section VI, we demonstrate that our technique performs significantly better than the other static approaches we discussed. Using the ILP provides an optimal solution for the given assumptions, and this way we can constitute a baseline for comparing dynamic scheduling approaches. Moreover, if the workload can be estimated *a priori*, static optimization can be utilized for achieving low and stable temperature profiles at runtime. However, in many systems, workload changes dynamically and online methods are required to manage temperature and reliability. To address this, we propose dynamic temperature-aware scheduling approaches in Section IV.

IV. Dynamic Temperature-Aware Scheduling

Workload characteristics typically vary dynamically during execution, making it hard to predict workload ahead of time especially in high-end computing domain. Thus, dynamic management of temperature is required to achieve higher system reliability and to reduce the design challenges caused by hot spots and temperature variations. To avoid high performance impact, temperature-aware scheduling has to be fast and easy to implement.

In this section, we present OS-level temperature-aware scheduling techniques with negligible performance overhead. These techniques mitigate the thermal hot spots and large temperature variations by taking into account the temperature measurements of the MPSoC and adapting to changes. When combined with previously introduced reactive thermal management methods such as thread migration [8] and voltage scaling [33], our techniques can achieve even lower and more stable thermal profiles while reducing the performance impact of reactive techniques significantly. The thermally-aware policies we investigate are cost-effective and can be easily implemented into existing schedulers at the OS level.

The techniques we propose make decisions based on the temperature measured on the MPSoC. Current chips typically contain several thermal sensors, and these sensors can be read by a Continuous System Telemetry Harness (CSTH) for collecting and analyzing time series sensor data [9].

We next discuss the multiprocessor schedulers in state-of-art operating systems. We then follow with two previously introduced reactive thermal management methods that are applicable to MPSoCs. Finally, we explain the dynamic temperature-aware scheduling techniques we propose.

A. State-of-the-Art Load Balancing Schedulers

Many modern OS schedulers are based on multilevel queuing, which mixes several elements such as priority, round-robin, and shortest-job-first scheduling principles. For performance reasons, some amount of load balancing is commonly integrated in the scheduler. In Linux 2.6, each processor in the multiprocessor system has a queue, and a task stays in a queue for cache affinity. Tasks are moved to different queues only when the load is unbalanced (i.e., when length of a queue is less than one fourth of another). Solaris migrates threads to other processors when a core becomes overloaded. The thread migration in Solaris is performed based on giving priority to locality, following the assumption that the threads on nearby cores share the same caches.

In this paper, we implemented a dynamic strategy (`DLB`) where the scheduler balances the workload by sending workload to the least busy processor at each interval. This dynamic load balancing strategy is in principal similar to load balancing performed by operating systems such as Solaris, which balances the workload in the processors' queues at regular intervals. We used an interval of 20 s in our experiments, which provided a balanced workload profile.

B. Thermal Management Techniques for MPSoCs

Several techniques have been proposed in the literature to control the thermal behavior of MPSoCs. Here we discuss two previously introduced methods for managing temperature. Both of these techniques are reactive, that is they are activated only when a critical temperature is reached.

Dynamic thread migration (DTM) is an MPSoC thermal management method that migrates threads from hot processors to cooler ones. For minimizing the performance impact of thread migration, Heat-and-Run proposed loading the cores as much as possible and migrating workload when critical temperature values are observed [8]. In our implementation of this technique, we migrate the thread from the hot processor to the coolest processor available at that moment. The threshold temperature for migration is set at 85 °C, which is considered a high temperature for our system. Similar temperature values are shown as critical for other CPUs as well [31].

Voltage scaling for thermal management (VSTM) performs dynamic voltage and frequency scaling when the temperature reaches the threshold [33]. This technique lowers the temperature on the hot cores by reducing power consumption. In our implementation, we assume two built-in voltage/frequency settings for each core. Normally all jobs run at full speed ($f_{\max}$). If a core reaches the critical temperature (85 °C), the frequency/voltage level of the particular core is reduced to the lower setting (f_{low}) until the current job terminates. In our experiments, we picked f_{low} as two thirds of $f_{\max}$. Lower frequency settings can be used as well; however, this would increase the performance cost.

C. Low-Overhead Temperature Aware Scheduling

The policies we propose have negligible overhead in comparison to the existing decision-making process in OS-level multiprocessor schedulers, and they can be implemented in the OS scheduler with minimal changes. We first discuss two heuristic methods that make scheduling decisions based on the current temperature and floorplan. We then present a more advanced approach that adjusts the likelihood of each processor receiving workload based on the temperature history.

Heuristic Techniques: Reactive methods for thermal management have to trade off performance to control temperature. The heuristics we investigate here avoid excessive heating of cores by making scheduling decisions based on the current temperature of the cores. The two heuristics we implemented are: 1) *coolest*, where for each ready job, the scheduler selects the coolest processor for allocation and 2) *coolest-FLP*, where the principle is same as (1), but in addition the scheduler gives priority to processors that have "idle" neighbors. Horizontal heat transfer plays an important role in determining the temperature, as discussed previously in [29] and in Section III. Coolest-FLP exploits the fact that a significant amount of heat transfer occurs among neighboring units on the die, and active processors with idle neighbors will result in lower and more evenly distributed temperatures on the MPSoC. This way, the spatial variations in temperature can also be reduced.

Random Policy with Temperature Aware Adaptation: The scheduling policies we have discussed so far have different strengths. Load balancing achieves better load distribution and higher performance. On the other hand, making scheduling decisions based on current temperature decreases the hot spots and temperature gradients. In order to achieve a balanced load distribution, avoid thermal problems and yet avoid introducing significant complexity to the scheduler, we developed a probabilistic policy called `Adaptive-Random`.

This policy updates probabilities of sending workload to cores at each interval based on an analysis of the temperature history on the chip. Taking the history into account provides the ability to allocate workload on units exposed to lower thermal stress or that are on cooler parts of the MPSoC. For example, in Fig. 2, we see three cores with the same current temperature, but different histories. Assuming all cores are idle, the *Coolest* policy would not differentiate among these three cases. `Adaptive-Random` favors *core 1* due to core 1's lower temperature average in the history window. Core 1 is indeed a better choice than the others, as the thermal history suggests Core 1 and its neighbors have been under lower thermal stress. This way, `Adaptive-Random` achieves a better load distribution for performance purposes.

In `Adaptive-Random`, the new probability value for each core is computed using (9) at each job arrival. In the equation, P_n is the new probability, P_o is the previous probability, and W is the weight. P_n values saturate at 0 and 1. In order to evaluate the thermal stress on each core, W is computed at regular intervals using a sliding window of temperature history. The thermal constant of our system is on the order a few hundred milliseconds, so we set the interval and sliding window lengths at 1 s in order to account for the rapid changes in temperature. As we

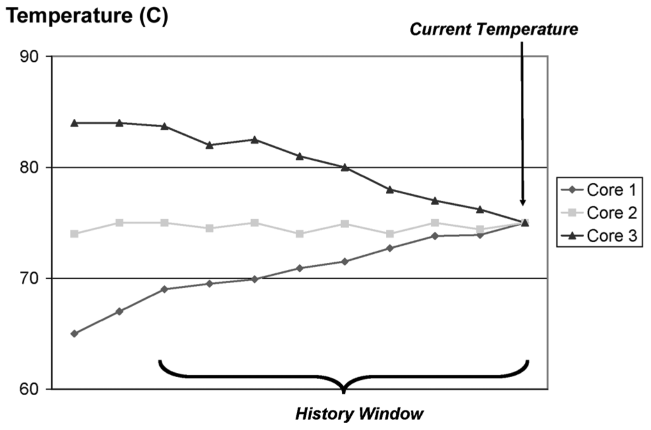

Fig. 2. Effect of temperature history.

compute only (9) at workload arrivals, the computation cost of our technique is negligible, and we do not have to stall execution. Once the probabilities are updated, the core for allocating the current job is selected through generating a random number

$$P_n = P_o \pm W. \tag{9}$$

The probability values are decremented or incremented by W_{dec} or W_{inc}, depending on whether the temperature has risen above the threshold temperature (T_{thr}), or dropped below a second threshold (T_{low}), respectively. In our simulations, we set T_{thr} at 80 °C. We used a threshold lower than 85 °C to prevent hot spots before they occur. In order to avoid allocating workload to cores that have temperatures slightly below 80 °C, we used a second threshold, T_{low}, set to 75 °C in our experiments. We do not increase P_n unless in the last interval the core temperature has dropped below T_{low}. Threshold values lower than 75 °C can introduce performance cost, as a number of cores would be idle until their temperatures are below the threshold. Moreover, setting the second threshold too low increases the temperature swing, which may accelerate the thermal cycling. Increasing the threshold value closer to 80 °C reduces its effect. For applying our technique to different systems, different threshold values can be selected following the same principles depending on the system and workload characteristics. There are three cases we consider while adjusting probability values.

1) If there are processors that have exceeded T_{thr} in the past interval, their P_n values are set to 0.
2) When the temperature of a core is between T_{thr} and T_{low}, no action is taken.
3) For cores that were below the second threshold T_{low} in the last interval, we increase their P_n by W_{inc} [see (10)]. While calculating W_{inc}, we evaluate Av_{thr}, which is the average temperature below T_{thr} divided by T_{thr}. This way, if a core is cooler than another, its W_{inc} is greater. We selected the β value as 0.1 empirically. We simulated β values between 0 and 0.5 at incrementation steps of 0.05, and selected the β with the best average case results for reducing hot spots and variations. For different MPSoCs, similar studies can be carried out to select the best β value

$$W_{\text{inc}} = \beta / Av_{\text{thr}}. \tag{10}$$

TABLE IV
POWER AND AREA DISTRIBUTIONS OF THE UNITS

Component Type	Power (%)	Area (%)
Cores	65.27	37.66
Caches	25.50	50.69
Crossbar	6.01	5.84
Other	3.22	5.81

This scheduling policy can be implemented on a real system easily with very low overhead. Collecting the thermal data and computation of weights do not impose noticeable performance impact. The random number generator can be implemented through a linear-feedback shift register (LFSR), which often already exists on the chip for test purposes. The rest of the computations (i.e., averages and ratios) are carried out incrementally throughout the execution. Another benefit of this policy is that it achieves better load balancing than making decisions solely on instantaneous temperature. The `Adaptive-Random` policy addresses the issues of maintaining a balanced and low temperature profile as well as distributing the thermal stress to cores as evenly as possible throughout system lifetime.

We next evaluate all the static and dynamic techniques we presented. The experimental methodology is explained in Section V and we provide the results in Section VI.

V. Experimental Methodology

In this section, we provide the details of the experimental setup. Our experimental results are based on the data collected from an UltraSPARC T1 processor, which contains eight cores and memory, communication, and input/output (I/O) units. This MPSoC has been manufactured in 90-nm process technology. The processors are in-order execution cores and have multithreading capability. In each core, four threads share an integer pipeline. Every two cores share an L2-cache and the cores communicate through shared memory. The power distribution among the units and relative sizes of each unit on the chip are provided in Table IV. The power data are updated values for those reported in [20], and they include the leakage estimates. UltraSPARC T1 (floorplan shown in Fig. 9) runs a multilevel queuing scheduler with basic load balancing capabilities as default.

We leverage continuous system telemetry harness (CSTH) [9] in our work to gather detailed workload characteristics of real applications. CSTH collects and analyzes real time data from hardware sensors, (e.g., currents, voltages, and temperatures) as well as software variables (e.g., performance metrics, memory accesses, etc.). CSTH runs as a part of the existing system software stack; therefore, the data processing does not introduce additional overhead.

To perform the thermal evaluation of the scheduling techniques we propose, we need to have a power consumption trace for each unit on the die. If we know when each unit is active or idle, we can estimate the instantaneous power consumption using the average power values. For SPARC cores, the peak power consumption is very close to the average power values [20]. Therefore, for cores, our goal is to determine when each core is active.

We sampled the utilization percentage for each hardware thread at every second using `mpstat` [23]. `mpstat` provides the distribution of user, kernel, and idle times. We recorded the utilization traces for half an hour for each benchmark.

To determine the active/idle time slots of cores more accurately, we used the kernel probes in `DTrace` [23] for recording the length of user and kernel threads. `DTrace` is a comprehensive dynamic tracing framework for Solaris. It should be noted that the lengths of threads we measure may not correspond to the overall execution time for a thread. For example, a thread might run for several minutes, but due to context switches, the continuous execution slices are of shorter length. Based on the length of the threads and the utilization traces obtained using `mpstat`, we reconstruct the workload trace.

We ran a set of benchmarks, which are grouped in four categories: 1) web server; 2) database applications; 3) commonly used integer benchmarks; and 4) multimedia benchmarks. For generating web server workload, we used SLAMD [34], which is a distributed application for load generation. The number of clients and threads can be tuned, allowing for simulating various workload intensity. We ran SLAMD with one client, and 20 and 40 threads per client to achieve medium and high utilization ratios, respectively.

To generate workload for database applications, we installed MySQL, and tested it with a multithreaded benchmarking tool, `sysbench`. We tested the MySQL database with various table sizes and number of threads. Using `sysbench`, we created a table with 1 million rows and 100 threads to access the database. For servers, combination of web and database applications are very commonly observed; therefore, we included the `Web&DB` application in our benchmark set as well. We also ran compiler (`gcc`) and compression/decompression (`gzip`). For the multimedia benchmarks, we ran MPlayer (integer) with a 640×272 sample video file. While simulating gcc and gzip, we ran six simultaneous copies of the application to increase the system utilization. While simulating MPlayer, we ran four instances of the video application for the same reason. We did not increase the number of applications further as these applications tend to become I/O and memory bound.

We summarize the details for our benchmarks in Table V. In Table V, we demonstrate the system utilization, which is averaged over all cores and throughout the execution. We also provide the maximum thread lengths measured. We gathered additional information for each benchmark using `cpustat`, such as cache misses and floating point instructions (see Table V). We use these characteristics to model the power consumption of the crossbar and floating point unit. We report the L2 cache misses, as these give an idea about how frequently the crossbar is accessed. The memory and floating point statistics are per 100-K instructions. In Table V, we compare the thermal profiles of the benchmarks as well. The "X" marks show the benchmarks that have high percentage (over 25%) of hot spots (i.e., HS) and high-magnitude thermal cycles (i.e., TC). The classification for spatial gradient profiles are similar to that of hot spots, so we report only one of them here. The benchmarks without "X" are

TABLE V
WORKLOAD CHARACTERISTICS

Benchmark	Core Util. (%)	Thread Length (ms)	L2 I Miss	L2 D Miss	FP inst.	Thermal	
						HS	TC
Web-med	53.12	134	12.9	167.7	31.2	X	X
Web-high	92.87	268	67.6	288.7	31.2	X	
Database	17.75	268	6.5	102.3	5.9		
Web & DB	75.12	536	21.5	115.3	24.1	X	X
gcc	15.25	268	31.7	96.2	18.1		
gzip	9	536	2	57	0.2		
MPlayer	6.5	268	9.6	136	1		
MPlayer &Web	26.62	134	9.1	66.8	29.9		X

prone to hot spots and variations for a low to medium percentage of time.

We implemented a simulator to fairly compare different scheduling techniques. In our simulation, we took representative traces collected at runtime for each workload category. For evaluating static approaches, based on these traces, we designed task graphs consisting of ten tasks for each benchmark that matched the utilization and task length characteristics. We simulated task graphs with and without task dependences. For dynamic approaches, we ran a half-hour trace collected for each benchmark with dynamically changing workload.

We solved the ILPs in our static method using lp_solve [22], which could solve an ILP for a set of ten tasks in 2 hours. In the first step of the simulator, the scheduler is given a list of jobs and their start times, which is provided by the ILP solution. The scheduler then applies a fixed scheduling strategy based on the ILP results. Thus, the performance overhead of this method during runtime is minimal. For dynamic approaches, the scheduler makes its decisions based on the arriving jobs and the current system state (e.g., temperature or load of cores, depending on the policy).

In the next step of the simulator, power values are derived based on each unit's execution profile. Dynamic power management or voltage scaling is also applied at this stage, depending on the policy simulated. For cores, we used average power values for the active and idle states for UltraSPARC T1. In the average case, the ratio between active and idle state power is 7.4. We estimated the dynamic power at the lower voltage levels based on the relationship between power, frequency and voltage (i.e., $P \propto f * V^2$). We assumed two built-in voltage/frequency settings in our simulations. To account for the leakage power, we used the second-order polynomial model proposed in [38]. This model computes the change in leakage power for the given differential temperature and voltage values. We determined the coefficients in the model empirically to match the normalized leakage values in [38]. This second-order model is shown to match closely with measurements. As we know the amount of leakage at the default voltage level for each core, we scaled it based on this model for each voltage level, taking both the temperature and voltage change into consideration. We used a sleep state power of 0.02 W, which is estimated based on sleep power of similar cores. For DPM, we implemented a fixed timeout policy [3] with timeout set to 100 ms. In addition, we investigate a combined policy of DPM and DVS. The hybrid DPM/DVS policy selects the lowest frequency possible for each task considering the deadline constraints, and shuts down the cores based on the fixed timeout policy. For the crossbar, we used a simple power model, where the power consumption scales according to how many cores are active and their L2 access characteristics derived from traces in Table V.

The next step is to obtain the temperature distributions using a thermal simulator. We used HotSpot version 2 [33] as the thermal modeling tool, and modified it accordingly our MPSoC. The thermal package characterization was based on the package properties of the UltraSPARC T1. We performed the thermal simulations using a sampling interval of 10 ms, which provided a good precision. We used steady-state temperature values obtained through HotSpot as the initial temperature.

Using the experimental methodology described before, we evaluated all the static and dynamic techniques we discussed. We provide the experimental results in Section VI.

VI. Experimental Results

In this section, we evaluate all of the scheduling techniques we discussed in Sections III and IV. We start by comparing the energy and temperature-aware static scheduling approaches introduced in Section III. We then provide results for the low-overhead dynamic scheduling techniques and compare them against the static baseline case. We evaluate the scheduling techniques by comparing their efficiency of reducing thermal hot spots, spatial gradients, and temporal fluctuations (i.e., thermal cycles). We show results for systems with DPM and DVS strategies to demonstrate how the schedulers perform when the system has power management capabilities.

The hot spot results show the percentage of time spent above 85 °C, which is considered a high temperature for our system. For Intel 1.5 GHz Pentium 4 processor and AMD 1.2 GHz Athlon processor, the recommended maximum die temperatures are 72 °C and 95 °C, respectively [31]. The spatial gradient results summarize the percentage of time gradients above 15°C are observed. Device delay is correlated with on-resistance, which increases with temperature. Gradients of even 15 °C–20°C start causing clock skew and delay issues [1]. The spatial gradient distribution is calculated by evaluating the temperature difference between hottest and coolest cores at each sampling interval.

We report the temporal fluctuations of magnitude above 20°C for only the cases with DPM and DVS/DPM, because going into sleep state causes large magnitude of variations in temperature. We do not provide thermal cycling results for the case of no power management due to the lack of significant temporal variations (i.e., the results are not available for this case in Table VI). The number of cycles to failure can be approximated using Coffin–Manson model [12]. For example, if we compute the failure rate for metallic structures, assuming the same frequency of cycles, when ΔT increases from 10 °C to 20 °C, failures happen 16 times more frequently. Thermal cycling results were obtained by computing the ΔT over a sliding window and averaging the ΔTs of all cores.

TABLE VI
SUMMARY OF EXPERIMENTAL RESULTS

	Thermal Hot Spots ($\% > 85^oC$)				Thermal Cycles ($\% > 20^oC$)				Spatial Gradients ($\% > 15^oC$)			
Benchmark	DLB	Ad.Rnd	Min-En	Min-Th&Sp	DLB	Ad.Rnd	Min-En	Min-Th&Sp	DLB	Ad.Rnd	Min-En	Min-Th&Sp
	No Power Management											
AVG	21.2	16.2	N/A	4.5	N/A	N/A	N/A	N/A	9.0	0	N/A	0.8
	DPM											
Web-med	27.2	20.5	8.5	7.5	36.6	21.8	12.3	6.5	17.0	2.9	9.4	2.1
Web-high	47.5	34.3	14.8	12.2	12.2	7.2	3.9	1.9	28.7	1.7	15.7	1.5
Database	9.7	7.5	2.8	0.0	22.3	10.3	7.7	3.0	6.2	1.4	3.6	1.2
Web&DB	38.4	26.7	12.0	10.6	29.5	15.0	10.3	3.5	23.8	1.3	13.1	1.1
gcc	7.2	6.6	2.5	0.0	20.3	12.3	5.8	1.9	4.6	0.1	2.5	0.0
gzip	4.4	5.0	1.5	0.0	12.0	7.3	4.8	1.5	2.8	0.0	1.5	0.0
MPlayer	3.0	3.3	1.0	0.0	9.5	6.0	3.9	1.5	2.6	0.0	1.7	0.0
MPlayer&Web	14.4	11.0	4.2	0.1	29.2	18.2	10.0	3.1	8.7	2.1	4.9	1.2
AVG	19.0	14.4	5.9	3.8	21.5	12.3	7.3	2.9	11.8	1.2	6.6	0.9
	DVS & DPM											
Web-med	20.4	13.7	4.8	5.1	22.0	10.8	6.6	3.5	13.4	2.5	7.5	1.2
Web-high	33.4	24.2	8.0	8.2	6.8	3.7	2.7	1.0	17.4	1.6	8.7	1.5
Database	7.3	4.0	2.1	0.0	12.9	5.3	4.0	2.1	4.0	1.3	2.3	1.1
Web&DB	26.1	18.3	5.8	6.8	17.1	7.3	5.7	2.7	15.6	1.2	7.9	1.0
gcc	5.1	5.4	1.5	0.0	11.4	6.1	3.4	1.9	4.4	0.0	2.3	0.0
gzip	3.4	3.9	0.8	0.0	7.3	3.8	2.4	1.2	2.3	0.0	1.2	0.0
MPlayer	2.2	2.7	0.6	0.0	5.7	2.9	2.0	0.9	2.6	0.0	1.6	0.0
MPlayer&Web	10.8	7.4	2.6	0.1	17.4	9.2	5.2	2.8	7.2	2.0	4.5	0.7
AVG	13.6	9.9	3.3	2.5	12.6	6.1	4.0	2.0	8.4	1.1	4.5	0.7

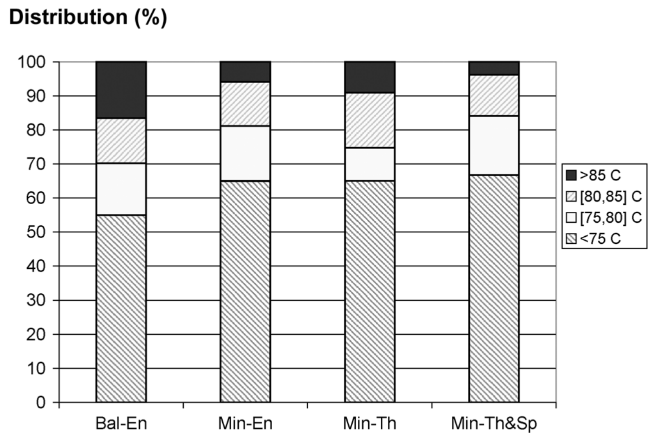

Fig. 3. Distribution of thermal hot spots, with DPM (ILP).

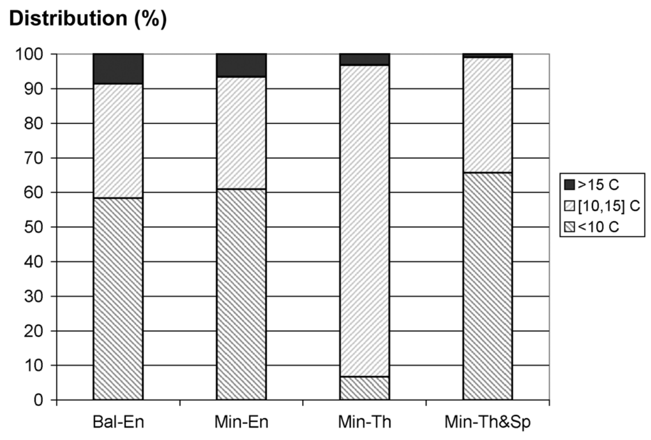

Fig. 4. Distribution of spatial gradients, with DPM (ILP).

A. Static Scheduling Techniques

We next provide an extensive comparison of the ILP based techniques. We refer to our static approach as `Min-Th&Sp`. As discussed in Section III, we implemented the ILP for minimizing thermal hot spots (`Min-Th`), energy balancing (`Bal-En`), and energy minimization (`Min-En`) to compare against our approach. To the best of our knowledge, this is the first time in the literature static MPSoC scheduling techniques are compared extensively to evaluate their thermal behavior.

We first show average results over all the benchmarks. Fig. 3 demonstrates the percentage of time spent at certain temperature intervals for the case with DPM. The figure shows that `Min-Th&Sp` achieves a higher reduction of hot spots in comparison to the other energy and temperature-based ILPs. The reason for this is that, avoiding clustering of workload in neighbor cores reduces the heating on the die, resulting in lower temperatures.

Fig. 4 shows the distribution of spatial gradients for the average case with DPM. In this plot, we can observe how `Min-Th` increases the percentage of high differentials while reducing hot spots. While `Min-Th` reduces the high spatial differentials above 15 °C, we observe a substantial increase in the spatial gradients above 10 °C. In contrast, our method achieves lower and more balanced temperature distribution in the die.

In Fig. 5, we show how the magnitudes of thermal cycles vary with the scheduling method. We demonstrate the average percentage of time the cores experience temporal variations of certain magnitudes. As can be observed in Fig. 5, `Min-Th&Sp` reduces the thermal cycles of magnitude 20 °C and higher significantly. The temporal fluctuations above 15 °C are reduced in comparison to other static techniques, except for `Min-En`. The cycles above 15 °C (total) occur 17.3% and 19.2% of the time for `Min-Th&Sp` and `Min-En`, respectively. Our formulation targets reducing the frequency of highest magnitude of hot spots and temperature variations, therefore, such slight increases with respect to `Min-En` are possible.

In the plots discussed before and also in Table VI, we observe that the `Min-Th&Sp` technique successfully reduces hot spots as well as the spatial and temporal fluctuations. Power

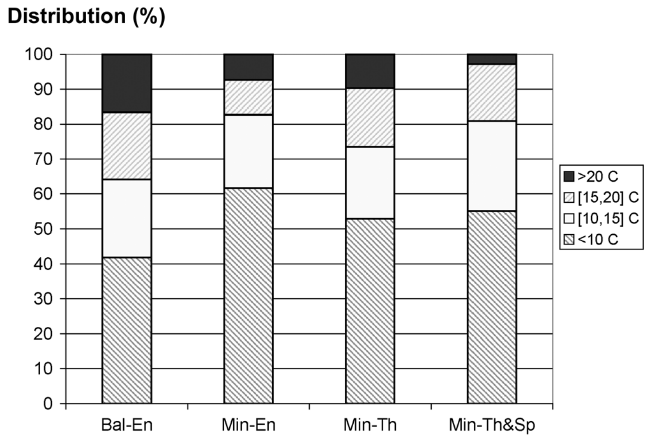

Fig. 5. Temporal variations, with DPM (ILP).

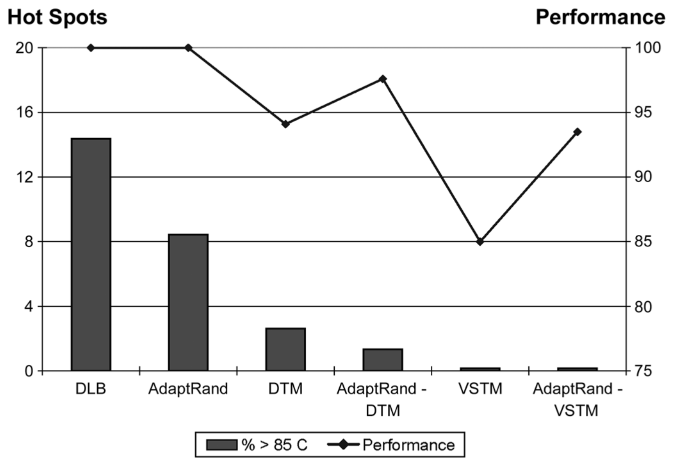

Fig. 7. Comparison of hot spots and performance cost

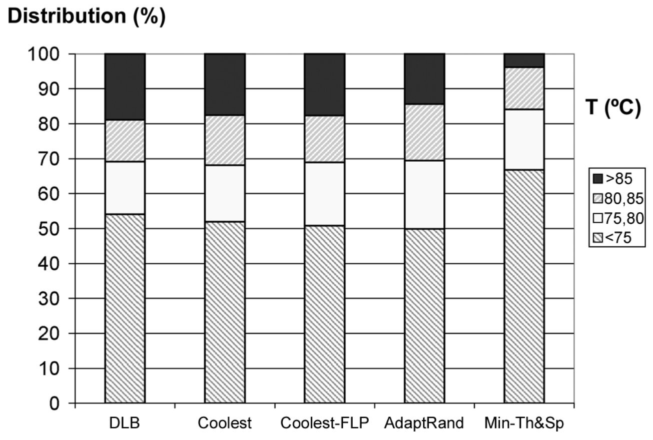

Fig. 6. Percentage of hot spots (dynamic techniques).

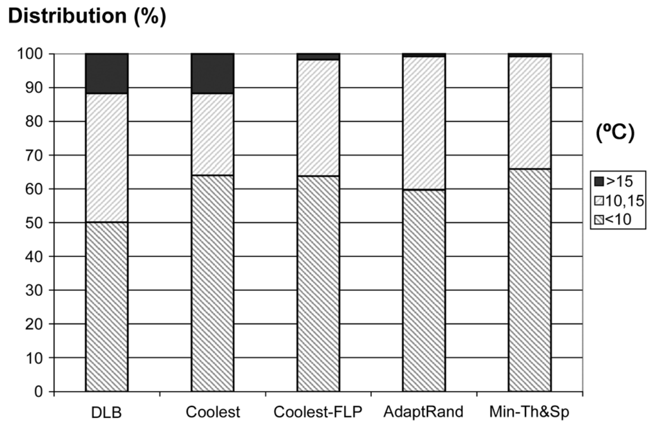

Fig. 8. Spatial gradients (dynamic techniques).

management (see DPM and DVS&DPM results for `Min-En` in Table VI) reduces the hot spots to some extent, but it cannot eliminate them effectively. Moreover, applying power management creates thermal cycles and larger spatial gradients due to the considerable decrease of power in sleep state. For example, `Bal-En` has high magnitude of cycles for 16% of the time (for DPM). `Min-En` reduces this percentage to about 7%. This reduction is due to the decrease in high temperatures. `Min-Th&Sp` can further decrease the frequency of cycles to less than 3%. We also observe that combining DVS with DPM reduces both high temperatures and temperature variations in comparison to applying only DPM.

The temperature balancing approaches `Min-Th` and `Min-Th&Sp` achieve much lower frequency of spatial gradients in comparison to energy-based techniques. For the cases with DVS&DPM, `Min-Th&Sp` bounds the frequency of spatial gradients to below 1.5% for all benchmarks except Web-high, which has over 90% utilization and a considerably high percentage of thermal hot spots.

`Min-Th&Sp` achieves more dramatic reductions in hot spots and gradients for benchmarks with lower system utilization (e.g., gcc and gzip), since the optimization method has more freedom to distribute the workload across the chip. As utilization increases (e.g., Web & DB and Web-high), we observe an increasing percentage of hot spots; however, the thermal cycles decrease as the system does not go into sleep state as often.

We have seen that our technique, `Min-Th&Sp`, outperforms other energy and temperature-based ILPs in terms of reducing both hot spots and temperature gradients. Minimizing energy (`Min-En`) reduces the hot spots due to the decrease in power, and manages to reduce gradients to some extent. However, by considering thermal profiles of tasks and the location of cores on the chip, `Min-Th&Sp` can achieve lower and more even temperature profiles.

Next, we evaluate the dynamic scheduling techniques. As our technique `Min-Th&Sp` performs the best among other static techniques in terms of reducing hot spots and variations, in Section VI-B, we only compare the dynamic approaches to `Min-Th&Sp`.

B. Dynamic Scheduling Techniques

In this section, we evaluate the dynamic scheduling techniques we discussed in Section IV. Fig. 6 shows the average case temperature distribution over time for dynamic cases, and also compares them against `Min-Th&Sp`. For example, load balancing results in temperatures over 85 °C for close to 20% of the time. `Adaptive-Random` can reduce the hot spots to around 14% without any performance impact.

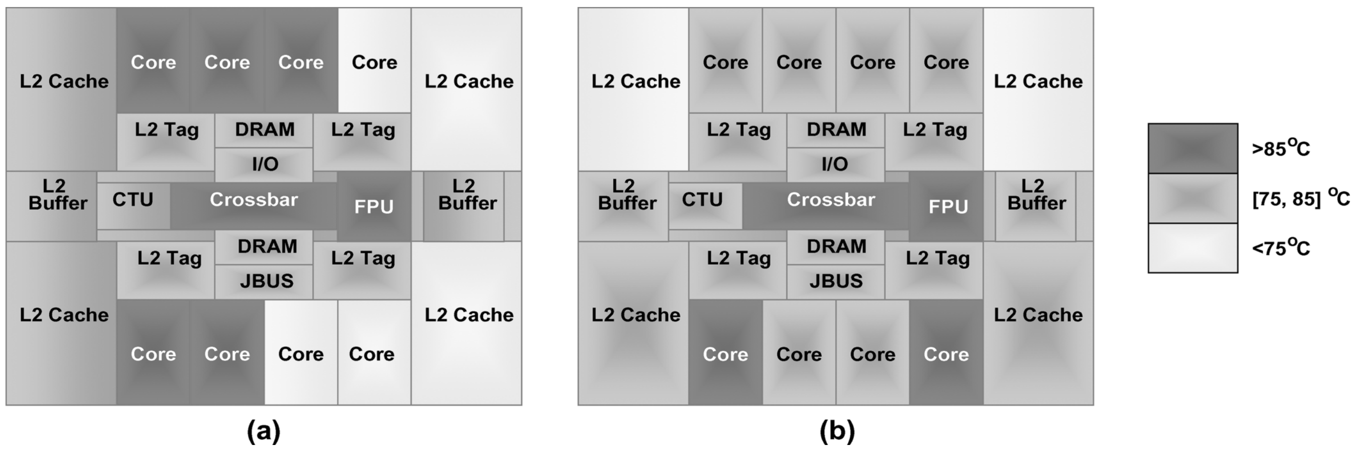

Fig. 9. Thermal maps. (a) DLB. (b) Adaptive-Random.

Fig. 7 demonstrates how effective the dynamic techniques are in reducing hot spots, and also compares the performance cost. The left axis provides the average percentage of time hot spots (over 85 °C) are observed, for the case without DPM. We show the normalized performance of each policy with respect to the baseline case of load balancing on the right axis. To evaluate the performance impact, we computed the average delay in the completion time of jobs with respect to the baseline case of load balancing. In thread migration, we assumed each migration takes 200 ms, according to the results provided in [4]. All the results regarding the Adaptive-Random technique are averaged over a hundred runs in order to obtain statistical convergence. While the aggressive reactive techniques (migration and voltage scaling) achieve reduced percentage of hot spots, their performance cost is very high. When these techniques are combined with `Adaptive-Random`, the hot spots can be further reduced to below 1%, while at the same time the performance degradation is reduced considerably, from 15% to below 7%.

In Fig. 8, we show how each dynamic technique effects spatial gradients. While Coolest-FLP results in very similar hot spot and thermal cycle percentages in comparison to Coolest, it reduces the frequency of high spatial gradients dramatically. This is due to the effect of heat sharing, as discussed previously in Section IV. Even though `Adaptive-Random` is a dynamic technique, it can almost eliminate the large gradients, and achieve very similar results to the baseline case of `Min-Th&Sp`.

Fig. 9 shows thermal maps of MPSoCs (a) and (b), which have been scheduled with `DLB` and `Adaptive-Random`, respectively. Load balancing causes high spatial temperature differences (above 20 °C) frequently, whereas `Adaptive-Random` typically achieves a more uniform thermal profile.

Fig. 10 compares the dynamic techniques to `Min-Th&Sp` in terms of the frequency of large magnitude of thermal cycles observed. `Adaptive-Random` can reduce the frequency of cycles that have magnitude greater than 20 °C by around 50%, with respect to load balancing.

We evaluate all the dynamic scheduling techniques and the reactive management techniques in Fig. 11 in terms of how they perform in reducing thermal cycles. Combining `Adaptive-Random` with thread migration or voltage scaling eliminates the high-magnitude cycles, at a much more reasonable performance cost than solely applying thread migration (DTM) or VSTM. A PI-controller based technique for temperature-aware DVS (such as [7]) can reduce the performance overhead of DVS. However, it should be noted that VSTM forces the cores to run at the lower voltage/frequency setting until the current job finishes, which

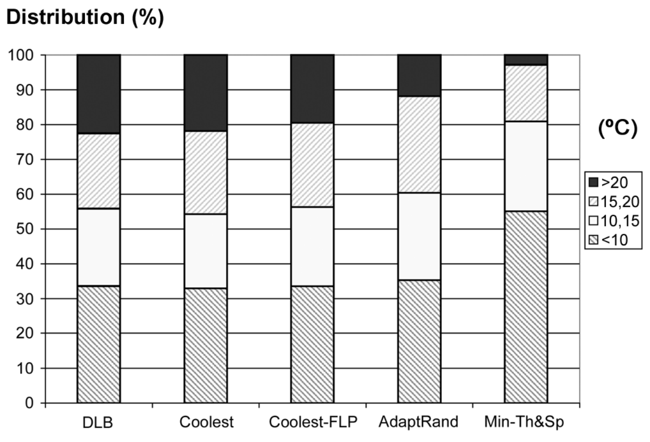

Fig. 10. Temporal variations, dynamic techniques (with DPM)

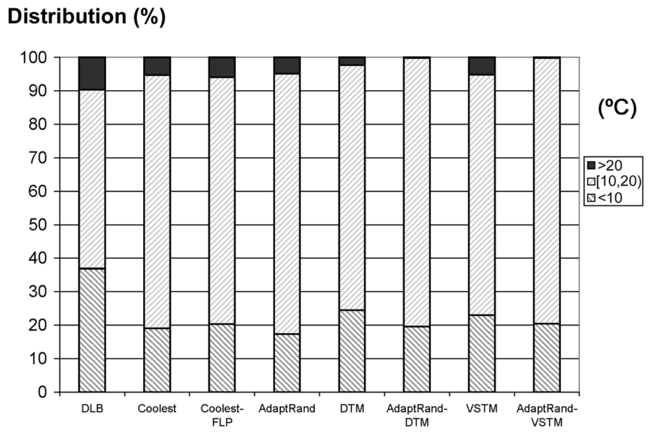

Fig. 11. Temporal variations (all dynamic techniques).

is expected to achieve lower temperatures with respect to the PI-controller approach.

Table VI shows a detailed comparison of load balancing (DLB), Adaptive-Random, `Min-En`, and `Min-Th&Sp`. We can observe that ILP methods achieve dramatic reductions in thermal hot spots, e.g., `Min-Th&Sp` decreases the percentage from 19 to around 4 for cases with DPM. Even though Adaptive-Random is an online technique, it can reduce the temporal fluctuations considerably, and performs very closely to `Min-Th&Sp` in terms of decreasing the frequency of high spatial gradients.

In this section, we have evaluated all the static and dynamic scheduling techniques we discussed. We showed that, our static temperature-aware optimization method `Min-Th&Sp` outperforms the other ILPs for reducing energy and hot spots in terms of reducing hot spots and large temperature gradients. Our solution reduces the frequency of hot spots by 35%, spatial gradients by 85% and thermal cycles by 61% in comparison to the ILP for minimizing energy. For dynamic management of temperature, we have demonstrated that our policy, `Adaptive-Random`, can reduce the temporal and spatial gradients by 50% and 90% respectively in comparison to state-of-the-art schedulers at negligible performance overhead.

VII. Conclusion

In this paper, we have investigated static and dynamic temperature-aware scheduling techniques for MPSoCs. We first proposed a static ILP-based optimization method to set a baseline for dynamic techniques. Our ILP minimizes the hot spots, as well as the temperature variations in time and space. We formulated ILPs for minimizing energy, balancing energy, and minimizing hot spots (without considering gradients), and provided an extensive comparison. We demonstrated that our static temperature-aware scheduling method outperforms the other ILP-based approaches in terms of reducing both hot spots and gradients, e.g., it reduces the frequency of hot spots by 35%, spatial gradients by 85% and thermal cycles by 61% in comparison to the ILP for minimizing energy.

We also developed an adaptive dynamic policy, which modifies the workload allocation policy based on the temperature history. While the adaptive method performed slightly better in eliminating hot spots than existing load balancing techniques, it provided 50% and 90% reductions in temporal and spatial temperature variations respectively. In addition, we demonstrated that the performance overhead of reactive techniques such as thread migration and voltage scaling can be reduced dramatically when they are combined with the Adaptive-Random technique, while achieving lower and more stable temperatures.

Acknowledgment

The authors would like to thank T. Marsh at Sun Microsystems for his help in setting up the experiments on UltraSPARC T1. They would also like to thank the anonymous reviewers for their valuable feedback.

References

[1] A. H. Ajami, K. Banerjee, and M. Pedram, “Modeling and analysis of nonuniform substrate temperature effects on global ULSI interconnects,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 24, no. 6, pp. 849–861, Jun. 2005.

[2] D. Atienza, P. Del Valle, G. Paci, F. Poletti, L. Benini, G. De Micheli, and J. M. Mendias, “A fast HW/SW FPGA-based thermal emulation framework for multi-processor system-on-chip,” in *Proc. DAC*, 2006, pp. 618–623.

[3] L. Benini, A. Bogliolo, and G. De Micheli, “A survey of design techniques for system-level dynamic power management,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 3, pp. 299–316, Jun. 2000.

[4] S. Bouchenak and D. Hagimont, “Pickling threads state in the java system,” in *Proc. Technol. Object-Oriented Languages Syst. (TOOLS)*, 2000, p. 22.

[5] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[6] K. Choi, R. Soma, and M. Pedram, “Dynamic voltage and frequency scaling based on workload decomposition,” in *Proc. ISLPED*, 2004, pp. 174–179.

[7] J. Donald and M. Martonosi, “Techniques for multicore thermal management: Classification and new exploration,” in *Proc. ISCA*, 2006, pp. 78–88.

[8] 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 *Proc. ASPLOS*, 2004, pp. 260–270.

[9] K. Gross, K. Whisnant, and A. Urmanov, “Electronic prognostics through continuous system telemetry,” in *Proc. MFPT*, 2006, pp. 53–62.

[10] J. Hu and R. Marculescu, “Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints,” in *Proc. DATE*, 2004, pp. 234–239.

[11] W.-L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, “Thermal-aware task allocation and scheduling for embedded systems,” in *Proc. DATE*, 2005, pp. 898–899.

[12] JEDEC Solid State Technology Association, Arlington, VA, “Failure mechanisms and models for semiconductor devices,” JEDEC publication JEP122C, 2006. [Online]. Available: http://www.jedec.org

[13] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, “Introduction to the cell multiprocessor,” *IBM J. Res. Development*, vol. 49, no. 4/5, pp. 589–604, Jul./Sep. 2005.

[14] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min, “Performance comparison of dynamic voltage scaling algorithms for hard real-time systems,” in *Proc. RTAS*, 2002, p. 219.

[15] P. Kongetira, K. Aingaran, and K. Olukotun, “Niagara: A 32-way multithreaded SPARC processor,” *IEEE Micro*, vol. 25, no. 2, pp. 21–29, Feb. 2005.

[16] H. Kufluoglu and M. A. Alam, “A computational model of NBTI and hot carrier injection time-exponents for MOSFET reliability,” *J. Computational Electron.*, vol. 3, no. 3, pp. 165–169, Oct. 2004.

[17] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha, “HybDTM: A coordinated hardware-software approach for dynamic thermal management,” in *Proc. DAC*, 2006, pp. 548–553.

[18] E. Kursun, C.-Y. Cher, A. Buyuktosunoglu, and P. Bose, “Investigating the effects of task scheduling on thermal behavior,” presented at the Proc. TACS, Boston, MA, 2006.

[19] C. J. Lasance, “Thermally driven reliability issues in microelectronic systems: Status-quo and challenges,” *Microelectron. Reliab.*, vol. 43, pp. 1969–1974, 2003.

[20] A. Leon, L. Jinuk, K. Tam, W. Bryg, F. Schumacher, P. Kongetira, D. Weisner, and A. Strong, “A power-efficient high-throughput 32-thread SPARC processor,” in *Proc. ISSCC*, 2006, p. 98.

[21] J. Liu, P. H. Chou, N. Bagherzadeh, and F. Kurdahi, “Power-aware scheduling under timing constraints for mission-critical embedded systems,” in *Proc. DAC*, 2001, pp. 840–845.

[22] “Lp_solve,” 2004. [Online]. Available: http://www.lpsolve.sourceforge.net/5.5/

[23] R. McDougall, J. Mauro, and B. Gregg, *Solaris Performance and Tools*. NJ: Sun Microsystems Press, 2006.

[24] M. Mutyam, F. Li, V. Narayanan, M. Kandemir, and M. J. Irwin, “Compiler-directed thermal management for VLIW functional units,” in *Proc. LCTES*, 2006, pp. 163–172.

[25] G. Quan and X. Hu, “Energy efficient fixed priority scheduling for real-time systems on variable voltage processors,” in *Proc. DAC*, 2001, pp. 828–833.

[26] P. Rong and M. Pedram, “Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system,” in *Proc. ASPDAC*, 2006, pp. 473–478.

[27] T. S. Rosing, K. Mihic, and G. De Micheli, “Power and reliability management of SoCs,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 4, pp. 391–403, Apr. 2007.

[28] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano, “Communication-aware allocation and scheduling framework for stream-oriented multi-processor system-on-chip,” in *Proc. DATE*, 2006, pp. 3–8.
[29] K. Sankaranarayanan, S. Velusamy, M. R. Stan, and K. Skadron, “A case for thermal-aware floorplanning at the microarchitectural level,” *J. Instruction-Level Parallelism*, vol. 7, pp. 1–16, 2005.
[30] M. Santarini, “Thermal integrity: A must for low-power IC digital design,” in *Proc. EDN*, Sep. 2005, pp. 37–42.
[31] Y.-H. Shih and J.-G. Hwu, “An on-chip temperature sensor by utilizing a MOS tunneling diode,” *IEEE Electron Device Lett.*, vol. 22, no. 6, pp. 299–301, Jun. 2001.
[32] K. Skadron, “Hybrid architectural dynamic thermal management,” in *Proc. DATE*, 2004, pp. 10–15.
[33] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, “Temperature-aware microarchitecture,” in *Proc. ISCA*, 2003, pp. 94–125.
[34] Sun Microsystems, “SLAMD distributed load engine,” [Online]. Available: www.slamd.com
[35] J. Srinivasan and S. V. Adve, “Predictive dynamic thermal management for multimedia applications,” in *Proc. ICS*, 2003, pp. 109–120.
[36] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, “The case for lifetime reliability-aware microprocessors,” in *Proc. ISCA*, 2004, p. 276.
[37] K. Stavrou and P. Trancoso, “Thermal-aware scheduling for future chip multiprocessors,” *EURASIP J. Embedded Syst.*, vol. 2007, p. 40, 2007.
[38] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif, “Full-chip leakage estimation considering power supply and temperature variations,” in *Proc. ISLPED*, 2003, pp. 78–83.
[39] V. V. Vazirani, *Approximation Algorithms*. Berlin, Germany: Springer, 2003.
[40] R. Viswanath, V. Wakharkar, A. Watwe, and V. Lebonheur, “Thermal performance challenges from silicon to systems,” *Intel Technol. J., Q3*, vol. 23, p. 16, 2000.
[41] Y. Yu and V. K. Prasanna, “Energy-balanced task allocation for collaborative processing in wireless sensor networks,” *Mobile Netw. Appl.*, vol. 10, pp. 115–131, 2005.
[42] Y. Zhang, X. S. Hu, and D. Z. Chen, “Task scheduling and voltage selection for energy minimization,” in *Proc. DAC*, 2002, pp. 183–188.

Ayşe Kıvılcım Coşkun received the B.S. degree in microelectronics engineering from Sabanci University, Turkey, in 2003, together with a minor degree in physics, and the M.S. degree in computer engineering from the University of California San Diego (UCSD), La Jolla, where she is currently pursuing the Ph.D. degree in computer science and engineering.

She has been an intern with Sun Microsystems in the System Dynamics Characterization and Control Team since June 2006. Her research interests include reliability and temperature modeling and optimization in multicore SoCs, and fault tolerant computer architectures. Her advisor at UCSD is T. Rosing. She worked as a summer intern with Ecole Polytechnique Fédérale de Lausanne (EPFL) in summer 2005, working with Dr. G. De Micheli and Dr. Y. Leblebici.

Tajana Šimunić Rosing received the M.S. degree in electrical engineering from University of Arizona, Tucson with a thesis topic on high-speed interconnect and driver-receiver circuit design, and the Ph.D. degree from Stanford University, Stanford, CA, in 2001, concurrently with finishing her Masters in Engineering Management. Her Ph.D. dissertation was entitled “Dynamic management of power consumption”.

She is currently an Assistant Professor with the Computer Science Department, University of California San Diego (UCSD), La Jolla. Her research interests include low-power system design, embedded systems, and wireless system design. Previously, she was a full time researcher with HP Labs while working part-time at Stanford University. At Stanford, she has been involved with leading research of a number of graduate students and has taught graduate level classes. Prior to pursuing her Ph.D., she worked as a Senior Design Engineer with Altera Corporation. She has served at a number of Technical Paper Committees, and has been an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS.

Keith A. Whisnant received the B.S., M.S., and Ph.D. degrees in computer engineering from the University of Illinois at Urbana-Champaign, Urbana-Champaign. His dissertation focused on the design of a reconfigurable runtime environment that provides a wide variety of software-implemented fault tolerance techniques to user applications.

He is a member of the System Dynamics Characterization and Control Team, Sun Microsystems. He is the chief architect for developing software around the group's research into telemetry, proactive fault monitoring, and electronic prognostics. Through a JPL fellowship, he applied his fault tolerance runtime environment towards protecting spaceborne parallel scientific applications. He was also involved in the design and implementation a fault injection software framework for validating and evaluating dependable systems.

Kenny C. Gross received the Ph.D. degree in nuclear engineering from the University of Cincinnati, Cincinnati, OH, in 1977.

He is a Distinguished Engineer for Sun Microsystems and is team leader for the System Dynamics Characterization and Control Team, Sun's Physical Sciences Research Center, San Diego. He specializes in advanced pattern recognition, continuous system telemetry, and dynamical system characterization for improving the reliability, availability, and serviceability of enterprise computing systems. He holds 167 U.S. patents issued and pending and has authored about 166 scientific publications.

Dr. Gross was a recipient of the 1998 R&D 100 Award for one of the top 100 technological innovations of that year, for an advanced statistical pattern recognition technique (MSET) that was originally developed for nuclear and aerospace applications and is now being used for a variety of applications to improve quality and availability for enterprise computer servers.