Temperature-Aware MPSoC Scheduling for Reducing Hot Spots and Gradients

Ayse Kivilcim Coskun[†] Tajana Simunic Rosing[†] Keith A. Whisnant[‡] Kenny C. Gross[‡]
[†]University of California, San Diego [‡]Sun Microsystems, San Diego

Abstract— **Thermal hot spots and temperature gradients on the die need to be minimized to manufacture reliable systems while meeting energy and performance constraints. In this work, we solve the task scheduling problem for multiprocessor system-on-chips (MPSoCs) using Integer Linear Programming (ILP). The goal of our optimization is minimizing the hot spots and balancing the temperature distribution on the die for a known set of tasks. Under the given assumptions about task characteristics, the solution is optimal. We compare our technique against optimal scheduling methods for energy minimization, energy balancing, and hot spot minimization, and show that our technique achieves significantly better thermal profiles. We also extend our technique to handle workload variations at runtime.**

I. Introduction

In deep-submicron era, thermal hot spots and large temperature gradients have brought significant challenges in reliability, performance, cooling costs and leakage power. In this work, we propose optimizing task scheduling to minimize both the thermal hot spots and the temperature variations in time and space. We solve task scheduling problems for minimizing energy, balancing energy, and minimizing hot spots only (i.e. without considering gradients) as well. We show that optimizing with energy constraints alone cannot eliminate the temperature induced problems; moreover, addressing temperature gradients results in significantly better temperature profiles in comparison to optimizing only for thermal hot spots. We also propose a hybrid scheduling approach that can adapt to workload changes at runtime.

In addition to increasing the cooling costs, thermal hot spots accelerate failure mechanisms such as electromigration, stress migration, and dielectric breakdown, which cause permanent device failures [9]. A $10-15^oC$ increase in operating temperature can result in a 2X decrease in the mean time to failure of the devices [25]. Leakage power is exponentially related to temperature, and the positive feedback loop between temperature and leakage can damage the circuit due to thermal runaway. High temperatures also degrade performance, as the effective operating speed of devices decreases with increasing temperature.

Previous work shows that addressing thermal hot spots alone is not enough to achieve high reliability, and temperature gradients determine device reliability at moderate temperatures [12]. High magnitude and frequency of thermal cycles (i.e. temporal fluctuations) cause accelerated package fatigue and plastic deformations of materials, and leads to permanent failures [9]. Temperature cycles are created by either low-frequency power changes (i.e. system power on/off), or workload rate changes and power management decisions, which happen much more frequently [19].

Spatial temperature variations across the chip can lead to performance degradation or logic failures. In process technologies below 0.13 μm, reliability issues arise due to negative bias temperature instability (NBTI) and hot carrier injection (HCI), as they cause the circuits to fail in meeting timing constraints [10]. Global clock networks are especially vulnerable to the design issues caused by 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^oC [1].

To date, temperature related problems have been addressed using techniques that lower the average temperature or keep the temperature under a given threshold. Power management (e.g. [17]) and dynamic thermal management (e.g. [20]) are such techniques. 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. Moreover, they do not focus on the effects of temperature variations, and typically introduce performance cost.

In this paper, we investigate how workload scheduling can be optimized in order to achieve temperature profiles that are beneficial for reliable MPSoC design. In contrast to thermal management techniques, which perform computation migration or clock gating (e.g. [20]) when temperature reaches critical values, our goal is to adjust the workload distribution to achieve the best temporal and spatial temperature distribution possible. We use integer linear programming (ILP) to formulate the temperature-aware scheduling problem. To the best of our knowledge, our work is the first to obtain a task schedule that meets real-time task deadlines while minimizing thermal hot spots and spatial temperature differentials across the die. We compare our technique against other energy and temperature-aware ILP-based scheduling techniques, and show that our method outperforms them. Our technique can be used as a baseline for developing thermally-aware dynamic scheduling strategies, or it can be implemented on systems where workload can be estimated *a priori*, such as some embedded systems. We also show how our solution can be extended to address runtime workload variations. Our experimental evaluation is based on realistic workload information collected through Sun's patented Continuous System Telemetry Harness (CSTH) [6] running on UltraSPARC T1 [13].

The rest of this paper is organized as follows. We discuss the related work in Section II. We describe our technique in Section III, and also explain how other energy and temperature based ILPs are formulated. In Section IV we provide the experimental methodology, and evaluate our technique. Section V concludes the paper.

II. Related Work

A number of strategies have been proposed to optimize scheduling with power and performance objectives. In [26], optimal voltage schedule on a core is computed using integer linear programming (ILP), and the approach is extended for multiprocessors using a heuristic for task allocation. A power management strategy for heterogeneous mission-critical MPSoCs is proposed in [14]. A static solution for scheduling concurrent communication and task execution for heterogeneous network-on-chips (NoCs) is presented in [7]. Rong et al. formulate an ILP to find the optimal voltage schedule for a system with a single core and peripheral devices, and also propose a three-phase solution framework [17]. In [18], the MPSoC scheduling problem is decomposed into allocation and scheduling sub-problems, which are solved using ILP and constraint programming, respectively, with the objectives of minimizing the data transfer and guaranteeing deadlines for the average case.

Thermal modeling and management methods have been proposed to address the temperature-induced challenges. HotSpot [20] is an

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

automated thermal model, which calculates the transient temperature response given the physical characteristics and power consumption of units on the die. A fast thermal emulation framework is introduced in [2], which reduces the simulation time considerably while maintaining accuracy. Thermal management methods either make runtime decisions to control temperature, or perform offline optimizations. Computation migration and fetch toggling are examples of dynamic thermal management techniques that keep the temperature below a critical threshold [20]. Heat-and-Run performs temperature-aware thread migration for multicore multithreaded systems [5]. In [3], the trade-offs between various hot spot mitigation schemes, thermal time constants and workload variations are investigated on a POWER5 system. Static methods for thermal and reliability management are based on system characterization at design time. Including temperature as a constraint in the co-synthesis framework and in task allocation is introduced in [8]. RAMP provides a reliability model at architecture level for temperature-related intrinsic hard failures [22]. In [19], a joint policy optimization for achieving a high amount of power savings while meeting the reliability criteria for MPSoCs is proposed.

In this work, we propose a temperature-aware MPSoC task scheduling methodology for reducing hot spots and minimizing temperature gradients, for a set of *a priori* known tasks with deadlines. Our work differs from [17], as we look into systems with multiple processing units and optimize the system for not only power but also for temperature. We show that optimizing for energy alone is not sufficient to eliminate the temperature induced problems, and addressing temperature gradients in the ILP results in significantly better temperature profiles in comparison to optimizing only for thermal hot spots. In order to adapt to dynamic workload changes, we propose a hybrid scheduling method, which outperforms dynamic scheduling policies for highly predictable workload.

III. Temperature-Aware Scheduling

In this section we describe a task scheduling approach for minimizing the frequency of both thermal hot spots and large temperature gradients. We formulate task scheduling for minimizing energy, balancing energy and minimizing the thermal hot spots (i.e. without considering gradients) to provide a comparison against our technique. We also extend our method to address dynamic workload variations.

Our goal is finding a task schedule for the MPSoC where the deadline and dependence constraints of tasks are met, while minimizing and balancing the temperature across the MPSoC. Addressing both the hot spots and gradients is our solution's (demonstrated as `Min-Th&Sp`) distinguishing feature from other energy and thermal based methods shown in Table I. We utilize integer linear programming (ILP), which has been used for solving multiprocessor scheduling problems with different objectives previously (e.g. [18]). The ILP solution provides optimal results for the given assumptions for task execution times, deadlines and temperature profiles. We show the objective functions of all the ILPs we solve in Table I. Next, we explain the ILP formulations in detail.

In our system and application model, we assume the MPSoC contains m processor units, $PU = PU_p;\ p = 1, ..., m$, and we model the applications using task graphs. In a graph $G = (T, E)$, each vertex represents a task ($T_i \in T$), which is a function or collection of functions. The edges ($E_{ij} \in E$) show the precedence constraints. We assume the deadlines (D_i) and worst-case execution times ($WCET_i$) of tasks are known *a priori*. In our model, each PU has v discrete voltage settings $V = V_k;\ k = 1, ..., v$ (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 $e_{ik} = (g_{active}(v_k) * t_i)$, where g_{active} is the function for power consumption and t_i is the execution time of T_i. Assuming the tasks execute up to their WCET and $WCET_i$ is given as the execution time in the default (highest) processor frequency, the execution time for T_i running at speed v_k is computed as: $t_i = WCET_i * (v_1/v_k)$.

The objective function of our ILP (`Min-Th&Sp`) has two parts: 1) Minimizing and balancing the thermal hot spots (H in Table I); 2) Minimizing the spatial gradients (G). (1) minimizes the maximum time spent above the threshold per core (max Q_p) to balance 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, which increase when several jobs are clustered in neighbor units while the rest of the units are idle. Contrarily, when the workload is spread out spatially across the die, more even temperature distributions are achieved due to the heat transfer from hot to cool cores. Therefore, avoiding scheduling tasks in neighboring cores at the same time, i.e. reducing the **overlap**, reduces the spatial gradients. Part (2) of the objective function (G in Table I) minimizes the total overlap. Minimizing the sum of H and G addresses both thermal hot spots and spatial gradients. The "time spent over a temperature threshold" metric has previously been used for profiling the thermal behavior and for evaluation in [11]. In some cases, a long time spent slightly past the threshold can result in higher reliability than a short time far past the threshold. However, as we minimize the *overlap* in addition to max Q_p, our method does not result in such abrupt temperature rises.

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 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 the objective function (H), to compute q_{ik} for each task, we perform thermal simulations. We initially assume that $q_{ik} = WCET_i$. We solve the ILP (`Min-Th&Sp`) for the given task graph, maintaining the deadlines and precedence constraints. We next perform thermal simulation, and record the time spent above the threshold temperature for each task. Then we insert these new q_{ik} estimates in the ILP, and solve the ILP again to get the final schedule. We set the threshold temperature to 85^oC in our simulations, as 85^oC is considered a high temperature for our system. To verify the accuracy of this q_{ik} estimation, we iterated the method until convergence for 5 randomly generated task sets of 10 tasks. We saw that the error in estimating q_{ik} values using only two iterations stayed typically below 5%.

In the second part of the objective function of `Min-Th&Sp` (G in Table III), we compute the total *overlap* in the schedule. Here, n_{pr} is equal to 1 only if cores p and r are adjacent to each other; and d_{ij} is equal to 1 only if the completion time of T_i is greater than the start time of T_j. The product $p_{ij}.d_{ij}$ equals 1 if T_i precedes T_j and if T_j starts before T_i finishes, which shows 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 constraints in the ILP. The x 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. We use two sets of precedence constraints. The first set, (d), makes sure the dependencies

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}; $EN_{total} = \{ \sum_{T_i \in T} \sum_{v_k} e_{ik} y_{ik}\} + I_{total}$; $I_{total} = \sum_{p \in PU} \{ \sum_{i,j \in T, i \neq j} x_{ip} x_{jp} m_{ij} \;\; idle(s_j - \tau_i)\}$

TABLE III. ILP FORMULATION FOR MIN-TH&SP

<table>
<tr><td colspan="2">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:</td></tr>
<tr><td>(a) $\forall T_i : \sum_{p} x_{ip} = 1$</td><td>Each task is assigned to only one PU</td></tr>
<tr><td>(b) $\forall T_i : \sum_{k} y_{ik} = 1$</td><td>Each task runs at only one V/f level</td></tr>
<tr><td>(c) $\tau_i = s_i + t_i$</td><td>Execution finish time for T_i</td></tr>
<tr><td>(d) $s_i \geq max_{E_{ji} \in E}\{\tau_j\}$</td><td>Task precedence</td></tr>
<tr><td>(e) $\tau_i \leq D_i$</td><td>Deadlines for all sink nodes</td></tr>
<tr><td>(f) $s_i \geq \tau_j;\ if\ p_{ji} = 1$</td><td>Precedence for tasks on the same core</td></tr>
<tr><td>(g) $p_{ij} + p_{ji} = 1$;
if $x_{ip} = x_{jp} = 1$</td><td>If T_i and T_j are scheduled on the same core, either T_i precedes T_j, or vice versa</td></tr>
</table>

are satisfied. In addition, 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 the deadlines are met.

`Min-Th&Sp` can be applied to systems that have dynamic power management (DPM) or dynamic voltage scaling (DVS). Without DVS, there is only one voltage setting, so $y_{i1} = 1$. For DPM, the q_{ik} estimates are derived through simulations with DPM, as putting cores into the sleep state affects the thermal behavior. For DVS cases, we assume that each task runs on a fixed frequency. We then evaluate the thermal profile of each task for all frequency/voltage levels. We next provide the details for the other ILP formulations 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 a threshold temperature for each core to minimize and balance the thermal hot spots. This ILP does not consider spatial gradients (i.e. G=0). The rest of the formulation is the same.

Energy balancing: In the ILP formulation for `Bal-En` the temperature variable q_{ik} in `Min-Th` is replaced with e_{ik}, which is the energy per task for running T_i at frequency v_k. Summing all the $e_{ik}y_{ik}$ terms computes the energy consumed per task.

Minimizing energy: For systems with DPM only, the ILP for `Min-En` is solved for only the default frequency. For DVS, the $\sum_{v_k} e_{ik}y_{ik}$ term computes the energy per task for the frequency level of task T_i. Also, the timing parameters (e.g. t_i, s_i, etc. in Table II) are computed considering the voltage/frequency settings of tasks. While computing the total energy EN_{total}, we consider the energy consumed during all active and idle periods. 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. I_{total} is the energy spent during all idle times and the *idle(y)* function computes the energy for individual idle time slots. When we apply a fixed timeout dynamic power management (DPM) strategy, the energy during the idle time *S* can be computed as below. Here, $e_{penalty}$ and $t_{penalty}$ are the energy and time overhead for switching into and out of the sleep state, respectively.

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

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

The ILP formulations discussed above include multiple nonlinear elements. Such problems can be linearized using standard techniques, and can then be solved by ILP solvers. When two integer variables (e.g. $x_{ip} \cdot x_{ir}$ in the equation for G in Table III) are multiplied, we define a new 0-1 variable X_{ipjr} with the constraints in Table IV (a). When multiplying a binary (1-0) variable with an integer value (e.g. $(p_{ij} \cdot s_i)$) we use the linearization in Table IV (b), where D is a suitably large bound for the variables.

TABLE IV. LINEARIZATION

(a) $X_{ipjr} = x_{ip} \cdot x_{jr}$	**(b)** $r_{ij} = p_{ij}.s_i$
$x_{ip} + x_{jr} - X_{ipjr} = 1$ $-x_{ip} - x_{jr} + 2X_{ipjr} = 0$	$r_{ij} - D \cdot p_{ij} = 0$ $-s_i + r_{ij} = 0$ $s_i - r_{ij} + D \cdot p_{ij} = D$

To linearize the step functions introduced by the d_{ij} variables (i.e. $d_{ij} = 1$ iff $\tau_i \geq s_j$), we use Equation 3. The multiplications in this equation are linearized as described before.

$$d_{ij}(t_i - s_j) + (1 - d_{ij})(-t_i + s_j) = 0 \qquad (3)$$

Thus, converting the nonlinear problem to a linear one is simple using these techniques. The problem size grows exponentially as the number of tasks (and/or voltage levels for the DVS case) increase. For large task sets, ILPs can be solved using LP relaxation and randomized rounding [24].

Hybrid Temperature-Aware Scheduling:

Our optimal scheduling technique schedules a task graph that is available *a priori*. Here we describe a hybrid static-dynamic policy to cover the cases where the workload deviates from the estimated task graph at runtime. For many multimedia and signal processing applications the workload is highly predictable at design time, with minimum runtime fluctuations. Typical server workloads vary over large time scales, e.g. several hours, thus creating opportunities for hybrid scheduling strategies.

As we have discussed previously, minimizing the *overlap* reduces hot spots and large gradients. Our hybrid method integrates `Min-Th&Sp` with a dynamic scheduling policy, `Coolest-FLP` [4], that exploits this principle to adapt to runtime variations. We assume core temperatures are available through thermal sensor readings. `Coolest-FLP` sends a task to the *coolest* core available, giving priority to the cores with idle neighbors to reduce the overlap.

The hybrid policy addresses the runtime variations as the following: **1) Variation in the number of tasks.** `Coolest-FLP` policy is applied for unexpected tasks, and the tasks in the original graph are allocated based on the ILP schedule. If there are missing tasks in the graph, `Coolest-FLP` is applied for all tasks until the end of the graph period. After the period is over, we switch back to the static schedule determined by the ILP. **2) Variation in execution times.** We do not modify the schedule if some tasks finish earlier. If there are tasks executing for a longer time than expected, we do not modify the allocation but the start times of the dependent tasks are increased by the amount of this unexpected execution time.

IV. Experimental Results

Our experimental results are based on the UltraSPARC T1 processor [13]. The average power consumption and area distribution of the units on the chip are provided in Table V, and the floorplan is available in [13]. The power data are updated values for those reported in [13], and they include the leakage estimates.

TABLE V. 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

We leveraged the Continuous System Telemetry Harness (CSTH) [6] to gather detailed workload characteristics of real applications. We sampled the utilization percentage for each hardware thread at every second using `mpstat` [16]. We recorded half an hour long traces for each benchmark. To determine the active/idle time slots of cores more accurately, we recorded the length of user and kernel threads using `DTrace` [16].

We ran the following sets of benchmarks: 1) Web server, 2) Database, 3) Common Integer, 4) Multimedia. To generate web server workload, we ran SLAMD [21] on one client with 20 and 40 threads per client to achieve medium and high utilization, respectively. For database applications, we tested MySQL using `sysbench` for a table with 1 million rows and 100 threads. We also ran compiler (`gcc`) and compression/decompression (`gzip`) benchmarks. For multimedia benchmarks, we ran mplayer (integer) with a 640x272 video file. We summarize the details of our benchmarks in Table VI. The utilization ratios are averaged over all cores and throughout the execution. Using `cpustat`, we recorded the cache misses and floating point (FP) instructions per 100K instructions to model the power consumption of the crossbar and FP unit.

In our simulation, we took representative traces of data collected for each workload category. Based on these traces, for each benchmark we designed task graphs consisting of 10 tasks that matched the characteristics. We simulated task graphs with and without task dependencies. For each task graph, we solved the ILP using lp_solve [15]. Our method applies a fixed scheduling strategy based on the task start times and allocation information obtained from the ILP results. Thus, the performance cost at runtime is minimal.

TABLE VI. Workload characteristics

Benchmark	Avg Util (%)	L2 I-Miss	L2 D-Miss	FP instr
Web-med	53.12	12.9	167.7	31.2
Web-high	92.87	67.6	288.7	31.2
Database	17.75	6.5	102.3	5.9
Web & DB	75.12	21.5	115.3	24.1
gcc	15.25	31.7	96.2	18.1
gzip	9	2	57	0.2
MPlayer	6.5	9.6	136	1
MPlayer&Web	26.62	9.1	66.8	29.9

Peak power consumption of SPARC is similar to the average power [13], so we assumed that the instantaneous power consumption is equal to the average power at each state (active, idle, sleep). In the average case, the ratio between active and idle state power for UltraSPARC T1 is 7.4. We estimated the power at lower voltage levels based on the relationship between power, frequency and voltage (i.e. $P \propto f * V^2$). We assumed three built-in voltage/frequency settings in our simulations. To account for the leakage power, we used the second-order polynomial model proposed in [23]. We determined the coefficients in the model empirically to match the normalized leakage values in the paper. 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, considering both the temperature and voltage change. We used a sleep state power of 0.02 Watts, which is estimated based on sleep power of similar cores. For DPM, we implemented a fixed timeout policy with timeout set to 100ms. We also investigated a combined DPM-DVS policy, which selects the lowest frequency possible for each task considering the deadlines, and shuts down the cores based on the timeout policy. For the crossbar, we used a simple power model, where the power consumption scales according to how many cores are active and the memory access characteristics.

We used HotSpot version 2 [20] as the thermal modeling tool, and modified the floorplan and thermal package characteristics for UltraSPARC T1. We performed the thermal simulations with a sampling interval of 10 ms, which provided a good precision. We initialized HotSpot with steady state temperature values.

Next, we evaluate our scheduling technique, and compare it against other static and dynamic scheduling methods. We refer to our technique as `Min-Th&Sp`. As discussed in Section III we implemented ILPs for minimizing thermal hot spots (`Min-Th`), energy balancing (`Bal-En`) and energy minimization (`Min-En`) to compare against `Min-Th&Sp`. We also implemented a dynamic load balancing strategy (`DLB`) that balances the workload by sending jobs to the least busy core in the current interval.

We evaluate our scheduling technique by comparing the 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.

Table VII provides average results for no power management, and detailed results for DPM and DPM/DVS for all the schedulers. The hot spot results show the percentage of time spent above 85^oC, which is considered a high temperature for our system. The spatial gradient results summarize the percentage of time that gradients above 15^oC occur, as gradients of $15 - 20^oC$ start causing clock skew and delay issues [1]. The spatial distribution is calculated by evaluating the temperature difference between hottest and coolest cores at each sampling interval. For metallic structures, assuming the same frequency of thermal cycles, when ΔT increases from 10 to 20^oC, failures happen 16 times more frequently [9]. So, we report the temporal fluctuations of magnitude above 20^oC. We discuss thermal cycling for only the cases with DPM and DVS/DPM, because going

TABLE VII. SUMMARY OF EXPERIMENTAL RESULTS

	Thermal Hot Spots($\% > 85^oC$)					Thermal Cycles ($\% > 20^oC$)					Spatial Gradients ($\% > 15^oC$)				
Benchmark	DLB	Bal -En	Min -En	Min -Th	Min-Th &Sp	DLB	Bal -En	Min -En	Min -Th	Min-Th &Sp	DLB	Bal -En	Min -En	Min -Th	Min-Th &Sp
	No Power Management														
AVG	21.2	18.8	N/A	10.5	4.5	N/A	N/A	N/A	N/A	N/A	9.0	7.1	N/A	2.9	0.8
	DPM														
Web-med	27.2	23.8	8.5	13.1	7.5	36.6	29.1	12.3	17.1	6.5	17.0	12.3	9.4	4.5	2.1
Web-high	47.5	41.3	14.8	22.7	12.2	12.2	8.8	3.9	4.4	1.9	28.7	20.7	15.7	7.2	1.7
Database	9.7	8.1	2.8	4.4	0.0	22.3	17.3	7.7	10.7	3.0	6.2	4.6	3.6	2.0	1.2
Web &DB	38.4	33.7	12.0	18.5	10.6	29.5	22.5	10.3	12.3	3.5	23.8	17.2	13.1	6.2	1.1
gcc	7.2	6.8	2.5	3.6	0.0	20.3	15.8	5.8	9.1	1.9	4.6	3.3	2.5	1.1	0.0
gzip	4.4	4.0	1.5	2.1	0.0	12.0	9.7	4.8	6.0	1.5	2.8	2.0	1.5	0.6	0.0
MPlayer	3.0	2.9	1.0	1.6	0.0	9.5	6.6	3.9	4.6	1.5	2.6	2.0	1.7	1.1	0.0
MPl &Web	14.4	11.9	4.2	6.6	0.1	31.4	23.2	10.0	13.7	3.1	8.7	6.4	4.9	2.5	1.3
AVG	19.0	16.6	5.9	9.1	3.8	21.5	16.6	7.3	9.7	2.9	11.8	8.6	6.5	3.1	0.9
	DVS & DPM														
Web-med	19.4	13.2	4.3	7.4	4.5	21.3	16.0	6.4	9.4	3.1	11.6	10.4	5.5	2.5	0.7
Web-high	32.9	22.4	7.3	12.7	7.4	6.5	4.8	1.9	2.8	1.0	16.7	14.0	8.7	4.3	1.3
Database	6.9	4.5	1.4	2.4	0.0	12.7	9.4	3.8	5.5	1.9	3.9	3.1	2.0	1.8	0.2
Web &DB	25.4	17.6	5.7	10.1	6.4	16.3	12.0	4.8	7.1	2.4	15.3	14.0	7.3	3.5	0.6
gcc	5.0	3.7	1.2	1.9	0.0	11.1	8.2	3.3	4.8	1.6	4.0	3.8	2.3	1.6	0.0
gzip	3.2	2.2	0.8	1.1	0.0	7.1	5.3	2.1	3.1	1.0	3.0	2.7	1.3	1.4	0.0
MPlayer	2.1	1.6	0.5	0.8	0.0	5.3	3.9	1.6	2.3	0.8	4.4	4.0	1.5	1.3	0.0
MPl &Web	10.1	6.5	2.1	3.2	0.1	17.0	12.5	5.0	7.4	2.5	6.9	6.9	4.2	1.2	0.4
AVG	13.1	9.0	2.9	4.9	2.3	12.1	9.0	3.6	5.3	1.8	8.2	7.4	4.1	2.2	0.4

into the sleep state causes large variations in temperature (i.e. we put N/A in the table for no power management). ΔT values we report are computed over a sliding window and averaged over all cores.

In Table VII, we see that Min-Th&Sp successfully reduces hot spots as well as the spatial and temporal fluctuations. Power management (see DPM and DVS&DPM results for Min-En) cannot eliminate hot spots as much as Min-Th&Sp. Moreover, DPM creates thermal cycles and larger spatial gradients due to the considerable decrease of power in the 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 further decreases 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 DPM alone. 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% for all benchmarks except Web-high, which has a high percentage of thermal hot spots.

Min-Th&Sp achieves dramatic reductions in hot spots and gradients for benchmarks with low system utilization (e.g. gcc and gzip), since there is more freedom to distribute the workload over 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 cores do not go into the sleep state as often.

We next show average results over all the benchmarks, comparing the dynamic, static and hybrid techniques. Figure 1 demonstrates the percentage of time spent between certain temperature intervals for the case with DPM. The figure shows that using ILP-based optimization provides significantly better results than load balancing. Min-Th&Sp achieves higher reduction of hot spots (i.e. over 85^oC) 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 chip, resulting in lower temperature across the die.

Figure 2 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^oC, we observe a substantial increase in the spatial gradients above 10^oC. In contrast, our method achieves lower and more balanced temperature distribution in the die.

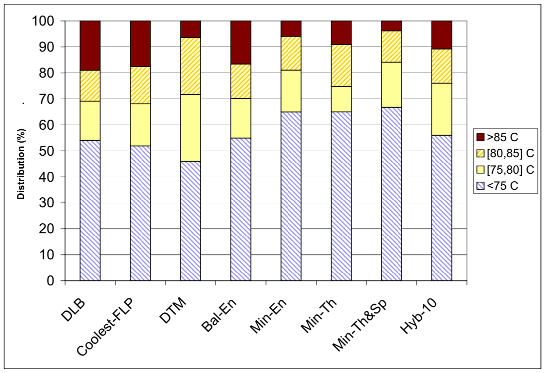

Fig. 1. Distribution of Thermal Hot Spots, with DPM

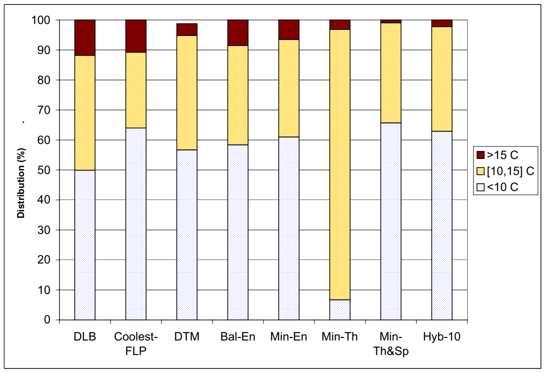

Fig. 2. Distribution of Spatial Gradients, with DPM

Finally, we show how thermal cycles are affected by the scheduling method. In Figure 3, we show the average percentage of time the cores experience temporal variations of certain magnitudes. Min-Th&Sp dramatically reduces the cycles of magnitude 20^oC and higher in comparison to other static and dynamic techniques.

We have seen that our technique, Min-Th&Sp, outperforms other energy and temperature based ILPs and dynamic load balancing

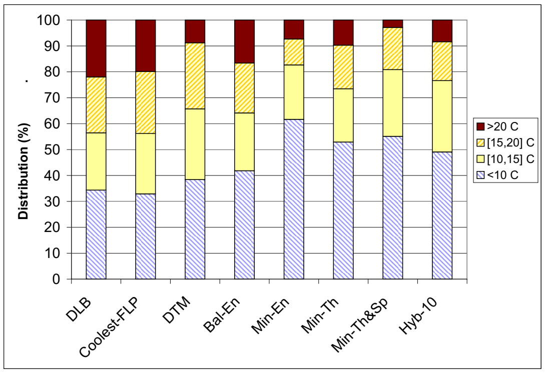

Fig. 3. Distribution of Temporal Variations, with DPM

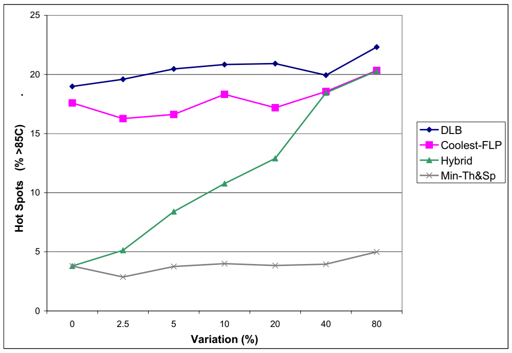

Fig. 4. Effect of hybrid scheduling for task graphs with variations

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. Our solution reduces the frequency of hot spots by 35%, spatial gradients by 85% and thermal cycles by 60% in comparison to the ILP for minimizing energy.

For evaluating our hybrid technique, we introduced random variations in task counts and execution times in the task graphs. In Figure 4, we compare our *hybrid* approach against Coolest-FLP alone, load balancing (DLB) and Min-Th&Sp. The x axis shows the percentage of variation (i.e. number of variations introduced divided by the original task count), and y axis shows the percentage of times we observe hot spots. As the amount of variation increase, the hybrid policy converges to the Coolest-FLP policy as expected. However, for highly predictable workloads with minor variations during runtime (such as multimedia and signal processing applications), we can see that it is significantly better to use the hybrid policy. In Figures 1, 2 and 3, we compare the hybrid policy for workload with 10% variation (Hyb-10) against other dynamic and static policies. We observe that (Hyb-10) achieves dramatic reduction in spatial gradients in comparison to Coolest-FLP and DLB; and for hot spots and cycles it brings around 40% improvement.

V. Conclusion

We presented an integer linear programming (ILP) based temperature-aware scheduling technique, which addresses both thermal hot spots and temperature gradients. In the experiments performed based on a real MPSoC, we compared our technique against dynamic load balancing, and optimal static solutions for energy minimization, energy balancing, and thermal balancing. We demonstrated that our technique successfully minimizes thermal hot spots, as well as spatial and temporal temperature fluctuations on the die, and achieves more reliable thermal profiles. We also extended our method to address workload variations at runtime. We showed that for highly predictable workloads, our hybrid policy achieves dramatic reductions in hot spots and gradients in comparison to dynamic techniques.

Acknowledgement

This work has been funded by Sun Microsystems, and the University of California MICRO grant 06-198. We thank Timothy Marsh at Sun Microsystems for his help in setting up the experiments on UltraSPARC T1. We also 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 Transactions on CAD*, 24(6):849–861, June 2005.

[2] D. Atienza, P. D. Valle, G. Paci, F. Poletti, L. Benini, G. D. Micheli, and J. M. Mendias. A fast HW/SW FPGA-based thermal emulation framework for multi-processor system-on-chip. In *DAC*, 2006.

[3] J. Choi, C.-Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose. Thermal-aware task scheduling at the system software level. In *ISLPED*, 2007.

[4] A. K. Coskun, T. Rosing, and K. Whisnant. Temperature aware task scheduling in MPSoCs. In *DATE*, 2007.

[5] 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 *ASPLOS*, 2004.

[6] K. Gross, K. Whisnant, and A. Urmanov. Electronic prognostics through continuous system telemetry. In *MFPT*, pages 53–62, April 2006.

[7] J. Hu and R. Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *DATE*, 2004.

[8] W.-L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Thermal-aware task allocation and scheduling for embedded systems. In *DATE*, 2005.

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

[10] H. Kufluoglu and M. A. Alam. A computational model of NBTI and hot carrier injection time-exponents for MOSFET reliability. *Journal of Computational Electronics*, 3 (3):165–169, Oct. 2004.

[11] E. Kursun, C.-Y. Cher, A. Buyuktosunoglu, and P.Bose. Investigating the effects of task scheduling on thermal behavior. In *TACS*, 2006.

[12] C. J. Lasance. Thermally driven reliability issues in microelectronic systems: status-quo and challenges. *Microelectronics Reliability*, 43:1969–1974, 2003.

[13] 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. *ISSCC*, 2006.

[14] J. Liu, P. H. Chou, N. Bagherzadeh, and F. Kurdahi. Power-aware scheduling under timing constraints for mission-critical embedded systems. In *DAC*, 2001.

[15] Lp_solve. http://lpsolve.sourceforge.net/5.5/.

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

[17] P. Rong and M. Pedram. Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system. In *ASPDAC*, 2006.

[18] 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 *DATE*, 2006.

[19] T. Simunic, K. Mihic, and G. D. Micheli. Optimization of reliability and power consumption in systems on a chip. In *PATMOS*, 2005.

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

[21] SLAMD Distributed Load Engine. www.slamd.com.

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

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

[24] V. V. Vazirani. Approximation algorithms. *Springer: Berlin, Germany*, 2003.

[25] R. Viswanath, V. Wakharkar, A. Watwe, and V. Lebonheur. Thermal performance challenges from silicon to systems. *Intel Technology Journal*, Q3, 2000.

[26] Y. Zhang, X. S. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *DAC*, 2002.