Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs

Thidapat Chantem, X. Sharon Hu, and Robert P. Dick

***Abstract*—Increasing integrated circuit (IC) power densities and temperatures may hamper multiprocessor system-on-chip (MPSoC) use in hard real-time systems. This article formalizes the temperature-aware real-time MPSoC assignment and scheduling problem and presents an optimal mixed integer linear programming based solution that considers the impact of scheduling and assignment decisions on MPSoC thermal profiles to directly minimize the chip peak temperature. We also introduce a flexible heuristic framework for task assignment and scheduling that permits system designers to trade off accuracy for running time to solve large problem instances. Finally, for task sets with sufficient slack, we show that inserting idle times between task executions can further reduce the peak temperature of the MPSoC quite significantly.**

I. Introduction

MULTIPROCESSOR systems-on-chips (MPSoCs) are now widely used in application-specific systems and high-performance computing. They offer performance, power consumption, and implementation complexity advantages over highly superscalar uniprocessor architectures. Their use, and scale, will increase dramatically in the coming years. According to Milchman [1], 16-core processors will be common within the next four years. Intel plans to deliver processors that have dozens or hundreds of cores during the next decade [2]. The use of heterogeneous MPSoCs can sometimes dramatically improve performance and power consumption relative to homogenous MPSoCs [3]. However, it can also increase complexity. It is likely that some future MPSoCs will be homogeneous and some will be heterogeneous. With the current use of MPSoCs in soft real-time applications such as gaming [4], it is expected that many hard real-time applications will soon be implemented using MPSoCs. In fact, FreeScale is now offering the QorIQ Embedded Multicore Processor [5], which is intended for, among others, aerospace applications, which contain hard real-time tasks.

MPSoC temperature is a strong function of power density. Increasing transistor counts and aggressive frequency scaling result in a significant increase in chip power density and temperature. Increasing chip temperature has significant impact on other design metrics including reliability, performance, and cost, as microprocessor failure rate depends exponentially upon operating temperature [6]. A 10–15 °C difference in operating temperature can result in a 2× difference in the lifespan of a device [7]. Temperature also affects speed; reduction of charge carrier mobility in transistors and increased interconnect latency resulting from high temperature degrade performance, requiring reduced clock frequencies or, worse yet, resulting in run-time failures.

Increasing power densities makes conservative package and cooling design prohibitively expensive, since the cost of cooling solutions increases super-linearly with power consumption [8]. It is therefore necessary to design packaging and cooling solutions based on less than worst-case thermal profiles and compensate by preventing, hopefully rare, dangerous thermal scenarios at run-time. Most popular approaches react to critical temperatures by reducing frequency and voltage (i.e., performing hardware throttling), or by temporarily preventing instruction issue to reduce the power consumption, and hence temperature, of the processor [9].

Since the execution times of real-time tasks, and hence total system utilization, tend to vary significantly due to factors such as conditional branches and system inputs [10], real-time applications can exhibit great temperature variation at run-time. When the system utilization is low, the MPSoC may not have a high temperature problem, thanks to the amount of slack available in the system. On the other hand, a system with high utilization can push an MPSoC to its thermal limit [11]–[13]. In the worst case, the host MPSoC may lack run-time thermal management, leading to overheating and signal timing violations or permanent failure. More subtly, real-time systems containing MPSoCs that support run-time thermal management would also fail when a temperature bound is reached, but for a different reason.

Most run-time thermal management techniques use thermal sensors to detect when the maximum safe temperature is approached and react by decreasing processor power consumption, e.g., by decreasing frequency or stalling instruction issue. These techniques share a common weakness: they decrease performance. If a real-time task running on an MPSoC with run-time thermal management ever triggers throttling when there is little timing slack, the real-time task will miss its deadlines. Missing hard real-time deadlines is unacceptable; an example would be failure to stop an automatically controlled train on time [14]. To guarantee hard real-time performance, designers should consider thermal effects by explicitly optimizing peak temperature while meeting all functionality and real-time deadlines. Clearly, the temperature-aware real-time MPSoC assignment and scheduling problem must be solved.

Existing power-aware techniques, such as energy minimization, peak power minimization, as well as global dynamic voltage and frequency scaling (DVFS), cannot solve the temperature problem in MPSoCs because they do not consider spatial thermal variation; heat generated by an active core also

T. Chantem and X.S. Hu are with the department of Computer Science & Engineering at the University of Notre Dame, Notre Dame, IN 46556. R.P. Dick is with the department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, MI 48109. Emails: {tchantem, shu}@nd.edu and dickrp@eecs.umich.edu. This work was supported in part by NSF under grant numbers CNS-0834180, CNS07-20457, CCF-0702761, and CNS-0347941 and by SRC under grant number 2007-HJ-1593.

affects other neighboring thermal elements, be they other cores or portions of the heatsink. The net heat flow from one thermal element to another depends on the conductance parameters and the current temperatures of these thermal elements. Ignoring spatial thermal variation can lead to unnecessarily-high peak temperatures, especially for high power density chips.

I.A. Related Work

Researchers have only recently started work in temperature-aware high-level synthesis [15] and design space exploration [16]. The objective is usually to optimize system performance subject to a peak temperature constraint. For uniprocessor architectures, Wang and Bettati presented a reactive two-speed policy to control the peak temperature [13]. To guarantee real-time deadlines, a proactive thermal management policy was later proposed [12]. Rao et al. presented an optimal processor speed control to maximize the work completed under a temperature constraint [17]. The thermal model was later improved by the same authors [18]. Mutapcic et al. focused on energy minimization under thermal and task constraints [19]. Quan et al. presented a necessary and sufficient condition for schedulability as well as a novel scheduling algorithm for real-time applications running on processors with a temperature constraint [20]. A temperature constraint may be sufficient if only timing faults are considered and if the designer has control over the post-deployment thermal environment, e.g., fan speed and ambient temperature. If wear is also considered, then lower temperatures are better. In addition, due to factors such as a high ambient temperature or a broken fan, the peak temperature minimization approach will maximize the number of systems that still meet the temperature constraint under severe conditions. Unfortunately, there is little research that targets peak temperature control directly. Bansal et al. were among the first to study the problem of peak temperature minimization using continuous dynamic speed scaling for uniprocessors running independent tasks [21]. Jayaseelan and Mitra presented a task sequencing technique to minimize the peak temperature for periodic real-time tasks running on a single processor [22]. Neither work considers MPSoCs nor task dependencies.

For multiprocessors and MPSoCs, the problem of assigning and scheduling real-time tasks is an important topic that has received significant research attention. Some papers focus on meeting hard real-time constraints [23] while others aim to optimize energy consumption in the presence of timing constraints [24]. Since most real-time scheduling problem variants are $\mathcal{NP}$-hard, many heuristics have been proposed to solve large problem instances with different optimality criterion [25]. Once again, the focus is usually placed on meeting the thermal constraint instead of minimizing peak temperature. For example, Rao et al. presented a method to maximize throughput by determining speeds of different cores subject to a peak temperature constraint [26]. Mulas et al. proposed a task migration algorithm that balances the loads on different cores to reduce hotspots [27]. Coskun et al. used online learning [28] and integer linear program (ILP) [29] to reduce the frequency of peak temperature constraint violations. Jung et al. used dynamic thermal management (DTM) to minimize energy while meeting a peak temperature constraint [30]. An approximation algorithm for minimizing the peak temperature of ideal processors was proposed by Chen et al. for real-time tasks with no precedence constraints [31]. In addition, a temperature-aware task assignment and voltage selection algorithm was proposed by Sun et al. for three-dimensional stacked-wafer MPSoCs [32]. However, this solution cannot be used to solve our problem since only homogeneous cores are considered and spatial thermal variation is ignored (the peak temperature of 3-D MPSoCs is strongly influenced by vertical inter-core heat flow).

Xie and Hung were the first to propose a collection of heuristics for temperature-aware processor allocation, task assignment, and scheduling [11]. However, the proposed heuristics either consider spatial or temporal thermal variations, but not both types of variations at the same time. In Section Section V, we show that their technique can deviate significantly from optimality.

Finally, Paci et al. claim that temperature-aware design is unnecessary in low-power embedded systems [33]. While their conclusions hold for very low-power embedded processors because on-die temperature variation is small, our results show substantial ($> 30\,^{\circ}\mathrm{C}$ improvement) benefits from temperature-aware design for MPSoCs containing processor cores characterized by the Embedded Systems Benchmarks Consortium [34] using the thermal model in Section II-B.

I.B. Contributions

This article makes the following main contributions. We present a mixed-integer linear programming (MILP) formulation for assigning and scheduling tasks with hard real-time constraints on an MPSoC to minimize the chip peak temperature. Our formulation considers spatial and temporal thermal variations. It relies on a *phased steady-state* thermal analysis directly integrated within the MILP formulation. The phased steady-state thermal analysis produces a separate steady-state thermal profile for each power profile occurring during the schedule. Extensions for temperature-dependent leakage power modeling, DVFS, finer-grained thermal modeling, and inter-task communication modeling are given.

To solve problem instances that are large or for which the effects of heat capacitance are significant, we propose a heuristic task assignment and scheduling framework in which the actual method for computing the thermal profile can be selected as appropriate. Specifically, phased steady-state thermal analysis is used when task durations are long relative to the time constants of the cores. *Transient thermal analysis*, in which temperatures are time-dependent, is used otherwise.

To exploit slack in the system where the effects of heat capacitance are significant, we use the concept of delay (i.e., idle time) insertion in our transient analysis based heuristic to further reduce the chip peak temperature while guaranteeing hard real-time deadlines.

I.C. Organization

The paper is organized as follows. In Section II, we introduce our system model, state our assumptions, and formally

define the problem. We motivate the need for a temperature-aware assignment and scheduling algorithm in Section III. We describe our formal approach in Section IV and present our flexible heuristic framework in Section V. We introduce and incorporate the concept of delay insertion into our heuristic framework in Section VI. The benefits and efficiency of our approach are experimentally determined in Section VII. Section VIII concludes the paper.

II. System Model and Problem Definition

The system model and the temperature-aware real-time MPSoC assignment and scheduling problem are now described.

II.A. Task Model

In our model, J represents the set of hard real-time tasks to be executed. For each task $j \in J$, the worst-case execution time when running on core m is denoted by $E(j,m)$, the deadline by $D(j)$, and the release time by $R(j)$. Note that $R(j) = 0$ and $D(j) = \infty$ if no release time and deadline constraints are associated with task j. A directed acyclic graph (DAG) is used to capture data dependencies among tasks. In a DAG, nodes represent tasks and directed edges indicate data dependencies between pairs of tasks. Let Γ_{j_1,j_2} denotes the dependency between tasks j_1 and j_2 where

$$\Gamma_{j_1,j_2} = \begin{cases} 1 & \text{if task } j_1 \text{ is immediately precedes task } j_2 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

A task j may execute only after all its predecessor tasks have completed and j has been released, i.e., the current time is at or later than $R(j)$. For now, we assume that there is no communication cost among dependent tasks. This assumption will be relaxed in Section IV-E. For periodic systems, we guarantee schedule validity by scheduling out to the hyperperiod of all tasks [35]. The *hyperperiod* is the least common multiple of the periods of all tasks in the problem specification.

II.B. Thermal Model

We model an MPSoC with a set of cores, M. For each core $m \in M$, its width, height, and location are specified. Based on the floorplan, the set of neighbors of core m, N_m, thermal conductance to a neighbor n, $G_n(m,n)$, and thermal conductance to the heatsink element above it, $G_h(m)$, can be calculated. For each task and core combination, $P(j,m)$ indicates the power consumption of core m when executing task j. We discuss an extension to this power model to account for leakage power in Section IV-B.

Thermal analysis estimates the heat transfer through heterogeneous materials among heat producers (e.g., transistors) and heat consumers (e.g., heatsinks attached to an MPSoC). In the task assignment and scheduling phase, we will adopt a coarse-grained discrete heat flow model analogous to widely-used compact models [36] to balance thermal analysis efficiency and accuracy. However, the algorithm framework proposed in Section V can be used with any thermal analysis technique.

In our thermal model, which is based on the classical Fourier heat flow model, each core corresponds to a discrete thermal element. (Section IV-D discusses how our approach can be modified to support finer-grained thermal element modeling.) The heatsink on top of the cores is modeled using multiple thermal elements and its partitioning corresponds to the layout of the cores. Since the heatsink is usually larger than the processor itself, we model heatsink overhang using additional thermal elements; the heatsink overhangs the chip by 25% of its length and width. The interface layer is included within the heatsink instead of being modeled explicitly. (The interface material is usually very thin so lateral heat flow within it can be neglected.) Lateral heat flow between cores and heatsink elements is modeled.

To perform thermal analysis, we take advantage of the well-known duality between electrical and thermal circuits. The temperature of each thermal element can be expressed as a function of its power consumption, the ambient temperature, and the temperatures of neighboring thermal elements. Figure 1(a) depicts the circuit representation of this model. Here, T_A denotes the ambient temperature, $G_A(h)$ is the conductance from the heatsink element h to the ambient, and $G_{nh}(h,g)$ is the conductance between heatsink elements h and g. The current source P_m denotes the power consumption of core m. The terms $G_h(m)$ and $G_n(m,n)$ were as defined previously.

The temperature of core m at time t, $T(t,m)$, can be determined using the node thermal analysis of the circuit in Figure 1(a):

$$0 = \sum_{n \in N_m} (T(t,m) - T(t,n)) \cdot G_n(m,n) + C(m) \cdot \frac{dT(t,m)}{dt} + (T(t,m) - T(t,h)) \cdot G_h(m) - \sum_{j \in J} \alpha(t,j,m) \cdot P(j,m) \tag{2}$$

$$0 = \sum_{g \in N_h} (T(t,h) - T(t,g)) \cdot G_{nh}(h,g) + C(h) \cdot \frac{dT(t,h)}{dt} + (T(t,h) - T(t,m)) \cdot G_h(m) + (T(t,h) - T_A) \cdot G_A(h), \tag{3}$$

where $T(t,h)$ is the temperature of the heatsink element h directly above core m at time t, $C(m)$ is the capacitance of core m, $C(h)$ is the capacitance of heatsink element h, and $\alpha(t,j,m) = 1$ if task j is active on core m at time t. In the above two equations, if the heatsink element h is a heatsink overhang element, then the term $(T_h - T_m) \cdot G_H(m) = 0$.

The thermal conductance of core m to the heatsink element h directly above it, $G_h(m)$, can be computed as shown by Serway [37]:

$$G_h(m) = \frac{Area_m}{R_{chip} \cdot Area_{chip}}, \tag{4}$$

where $Area_m$ denotes the area of core m, $Area_{chip}$ represents the area of the chip, and $R_{chip} = \frac{th_{si}}{K_{si} \cdot Area_{chip}}$, th_{si} is the thickness of silicon, and K_{si} denotes its thermal conductivity. In our experiments, we set th_{si} and K_{si} to be 0.6 mm and 148 W/mK, respectively.

The conductance of a heatsink element h to the ambient can be calculated in a similar manner. That is, we substitute $Area_m$ and $Area_{chip}$ in Eq. 4 by the area of the heatsink element under consideration and the area of the entire heatsink, respectively. In addition, we replace R_{chip} with R_{HS} in Eq. 4, where $R_{HS} = \frac{T_{active} - T_{ambient}}{P_{chip}} - R_{chip}$, P_{chip} being the total power consumption of the chip and T_{active} the average active layer temperature when all cores are busy and $T_{ambient}$ is the ambient temperature. We set T_{active} and $T_{ambient}$ to 90 °C and 45 °C, respectively.

We compute the conductance between core m and its neighbor core n as follows.

$$G_n(m,n) = \frac{w_{mn} \cdot th_{si} \cdot K_{si}}{L_{mn}}, \tag{5}$$

where w_{mn} is the length of intersection between cores m and n, L_{mn} is the distance between the midpoint of m and that of n. The lateral conductance between two heatsink elements can be computed in a similar fashion. We assume that the heatsink is made of copper, with a thickness of 1 mm and thermal conductivity of 400 W/mK.

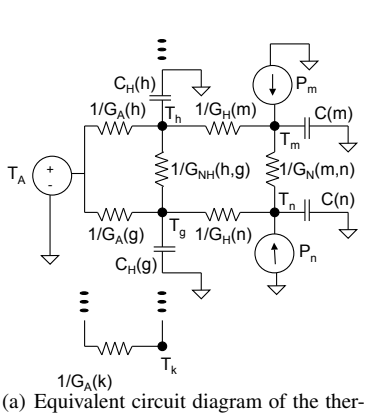
II.C. Problem Definition

Consider the floorplan of a chip containing a set of cores, M, and a set of hard real-time tasks, J as described above, determine a static assignment of tasks to cores and a static, non-preemptive schedule of tasks on the cores such that all precedence constraints and real-time deadlines are met, and the chip peak temperature, T_{max}, is minimized.

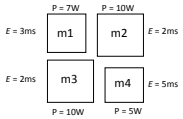
III. Motivating Example

Since average power (i.e., energy) for a fixed duration and peak power are related to chip temperature, it is natural to question whether optimizing peak temperature can produce significantly different results than optimizing peak power or average power. Let us consider a task set containing two identical tasks, j_1 and j_2, each with a deadline of 5 ms. For this example, the MPSoC is arranged as shown in Figure 1(b) (core sizes are not necessarily drawn to scale in the diagram). Task execution times (E) and associated power consumptions (P) are shown near the respective cores. To minimize energy, tasks j_1 and j_2 are both assigned to core $m2$. The resultant chip peak temperature is 65.30 °C. If our objective were to minimize peak power, then task j_1 would be assigned to core $m2$ and task j_2 to core $m1$, also resulting in a peak temperature of 65.30 °C. However, if task j_1 were executed on core $m4$ and task j_2 on core $m1$, the peak temperature would be reduced to 65.16 °C, which is about 0.14 °C cooler. This difference is the first point in the plot in Figure 2.

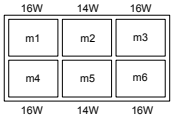
While the improvement in this case is small, the power density of the chip in the above example is only 0.19 W/mm^2. The power density can be as high as 0.79 W/mm^2 for 90 nm processors, 2.02 W/mm^2 for 65 nm processors, and 7.24 W/mm^2 for 45 nm processors [38]. To obtain similar chip power densities, we repeated the previous experiment in which the setup is explained earlier but increased each core power consumption by factors of 2, 5, 10, 15, and 20. The resulting chip power densities are 0.39 W/mm^2, 0.97 W/mm^2, 1.95 W/mm^2, 2.92 W/mm^2, and 3.89 W/mm^2. In each case, the heatsink conductance parameters to the ambient are adjusted to model improved cooling solutions necessary to maintain an average active layer temperature below the temperature constraint.



(a) Equivalent circuit diagram of the thermal model



(b) Floorplan for the motivating example with task execution time (E) and power consumption (P) for each core. Diagram not to scale



(c) A 2×3 floorplan

Fig. 1. Circuit and floorplans

Although task assignments and schedules are the same as before, chip peak temperatures increase when the higher power density cores are used. Figure 2 shows the reductions in chip peak temperatures when the peak temperature is optimized instead of peak power or energy for the example in Figure 1(b) and different chip power densities mentioned above. The x-axis shows the chip power density (in terms of factors mentioned previously). The y-axis shows the difference between the peak temperature obtained from energy or peak power minimization and that from peak temperature minimization. As can be seen from the plot, the advantages of minimizing the chip peak temperature directly increase with increasing chip power density, resulting in up to 20 °C reduction in peak temperature for this example.

Energy and peak power minimization suffer from the same weakness: neither considers spatial thermal effects. In fact, energy minimization ignores both temporal and spatial thermal variation while peak power minimization considers temporal thermal variation but ignore spatial thermal variation. The peak temperature of an MPSoC is increased by crowding the same amount of energy consumption into less time and space. Hence, to minimize the chip peak temperature, tasks should be assigned and scheduled in careful consideration of thermal interaction with neighboring cores. In addition, our example indicates that although there are many cases in which minimizing peak power produces different (and potentially better) results than minimizing energy, the same results are produced for some problem instances.

IV. MILP-based Approach

In this section, we present our formal approach in solving the problem defined in Section II-C. We also describe how

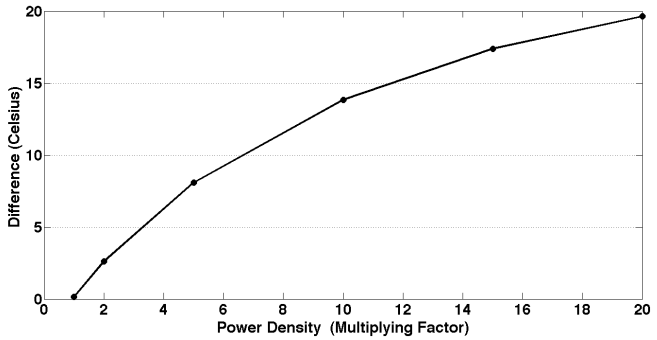


Fig. 2. Differences in peak temperatures: peak temperature minimization vs. peak power and energy minimization

our model can be extended to account for leakage power, dynamic voltage and frequency scaling (DVFS), and inter-task communication, as well as to include a finer-grained thermal model. A discussion on some limitations of the MILP-based approach is given at the end of the section.

IV.A. MILP Formulation

We now present our MILP formulation for the problem defined in Section II-C. We begin by defining the following variables.

$$\delta(j,m) = \begin{cases} 1 & \text{if task } j \text{ is assigned to core } m \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

$$\eta(j_1,j_2) = \begin{cases} 1 & \text{if task } j_1 \text{ starts before task } j_2 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

$$\beta(j_1,j_2,m) = \begin{cases} 1 & \text{if task } j_2 \text{ executes on core } m, \text{ precedes}^1, \\ & \text{and overlaps with task } j_1 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

To enforce consistency, we let

$$\beta(j,j,m) \equiv \delta(j,m). \tag{9}$$

The $\beta(j_1,j_2,m)$ variables capture the overlapping execution of different tasks and play a key role in computing the peak temperature. They are also useful in computing the peak power as will be shown later. We use $ts(j)$ and $tf(j)$ to denote the start and finish time of task j, respectively, yielding

$$tf(j) \equiv ts(j) + \sum_{m \in M} \delta(j,m) \cdot E(j,m) \tag{10}$$

$$\equiv ts(j) + et(j). \tag{11}$$

MILP formulations have long been proposed for modeling the task assignment and scheduling problem in a heterogeneous multiprocessor environment [39]. However, energy minimization has often been the main objective. Such solutions ignore both temporal and spatial thermal variation. Even peak power minimization only considers temporal thermal variation. To take both types of thermal variation into account, we directly minimize the chip peak temperature, T_{max}, which is the highest temperature at any position on the chip during a schedule of duration *SL*, i.e.,

$$T_{max} = \max_{m \in M, t \in [0,SL]} \tau_m(t), \tag{12}$$

Using Eq. 2 and Eq. 3 to compute the temperature at each node at any given time corresponds to dynamic or transient thermal analysis. Unfortunately, transient thermal analysis is computationally expensive. This makes the use of transient thermal analysis in the MILP formulation impractical; the MILP would only be able to solve very small problem instances, thereby making it difficult to validate a heuristic. For this reason, we set the capacitance values in Eq. 2 and Eq. 3 to zero to obtain the steady-state temperature at each node when predicting temperatures in our MILP formulation. In Section IV-F, we indicate the situations in which the MILP-based approach with steady-state analysis is appropriate and inappropriate. In addition, we present a solution to the more general problem of dynamic temperature optimization in Section V.

From the thermal model in Section II-B, it might appear necessary to compute the steady-state temperature, $\tau_m(t)$, of a core m at every time instant t to determine T_{max}. Even if we discretize the time duration SL, this approach may still be too costly; task execution times can vary dramatically, resulting in some tasks executing for hundreds of thousands or millions of time units. To overcome this difficulty, we make the following observations: 1) core power consumptions only change at the beginning or end of a task execution, and 2) the steady-state temperature of a core only experiences a rapid change when the power consumption of at least one core on the chip changes. Hence, we can significantly reduce the number of computations needed to obtain T_{max}. Specifically, we only evaluate the temperature of each core m immediately after every task i starts or finishes executing on any core in the MPSoC and denote this temperature by $T(i,m)$. Consequently, the objective function of the MILP can be expressed as

$$\min T_{max}, \text{ where } T_{max} \geq T(i,m), \forall m \in M, \forall i \in J. \tag{13}$$

$T(i,m)$ satisfies the constraints given in Eq. 2 and Eq. 3, which are rewritten in Eq. 14 and Eq. 15, respectively. Note that the capacitance values are set to zero: steady-state analysis is used.

$$T(i,m) \equiv T_{HS}(i,h) + \frac{1}{G_H(m)} \left[\sum_{j \in J} \beta(i,j,m) \cdot P(j,m) \right] + \frac{1}{G_H(m)} \sum_{n \in N_m} G_N(m,n) \cdot [T(i,n) - T(i,m)] \tag{14}$$

$$0 = (T_{HS}(i,h) - T(i,m)) \cdot G_H(m) + (T_{HS}(i,h) - T_A) \cdot G_A(h) + \sum_{g \in N_h} (T_{HS}(i,h) - T_{HS}(i,g)) \cdot G_{NH}(h,g). \tag{15}$$

Note that Eq. 14 is only linear if we can treat $P(j,m)$ as a constant given task j and core m. For now, we assume that

[1]Precedence is not necessary but is sufficient and simplifies the test.

this is the case. We will discuss the more general case where $P(j,m)$ is not a constant in Section IV-B.

The following constraints are used to guarantee schedulability.

1) Every task j is assigned to exactly one core m:

$$\forall j \in J \qquad \sum_{m \in M} \delta(j,m) = 1 \tag{16}$$

2) Every task j meets its deadline:

$$\forall j \in J \qquad ts(j) + et(j) \leq d(j) \tag{17}$$

3) Precedence constraints are honored:

$$\forall i,j \in J \qquad ts(j) \geq tf(i) \cdot \Gamma_{i,j} \tag{18}$$

4) All tasks execute for their durations without overlap:

$$\forall j_1, j_2 \in J$$
$$1 \leq \eta(j_1,j_2) + \eta(j_2,j_1) \tag{19}$$
$$ts(j_1) \leq ts(j_2) + (1 - \eta(j_1,j_2)) \cdot \Lambda \tag{20}$$
$$ts(j_2) \leq ts(j_1) + \eta(j_1,j_2) \cdot \Lambda \tag{21}$$
$$\forall j_1, j_2 \in J, j_1 \neq j_2, \forall m \in M$$
$$tf(j_1) \leq (2 - \delta(j_1,m) - \delta(j_2,m)) \cdot \Lambda + ts(j_2) + \Lambda \cdot (1 - \eta(j_1,j_2)) \tag{22}$$
$$tf(j_2) \leq (2 - \delta(j_1,m) - \delta(j_2,m)) \cdot \Lambda + ts(j_1) + \Lambda \cdot \eta(j_1,j_2), \tag{23}$$

where Λ is a constant greater than or equal to the largest deadline in the task set. Item 20 states that task j_1 must start before task j_2 if $\eta(j_1,j_2) = 1$. Item 22 guarantees that task j_1 finishes before task j_2 starts if tasks j_1 and j_2 are executed on the same processor and task j_1 precedes task j_2. Similar conditions hold for Item 21 and Item 23.

Consider a situation where tasks i and j execute on cores m and n, respectively. Further, task i precedes task j and their executions overlap. At the start of task i, we only need to consider the power consumption of core m. However, at the start of task j, we must take into account the power consumptions of both cores to correctly compute the chip peak temperature. For this reason, we must ensure that $\beta(j_1,j_2,m) = 1$ only when $\delta(j_2,m) = 1$ and $ts(j_2) \leq ts(j_1) \leq tf(j_2) - \epsilon$, where ϵ is a small constant used to prevent imprecise floating point computations from making it appear as if contiguous tasks overlap in time. Therefore,

$$\forall m \in M, \forall j_1, j_2 \in J, j_1 \neq j_2$$
$$tf(j_2) \geq ts(j_1) + (\beta(j_1,j_2,m) - 1) \cdot \Lambda \tag{24}$$
$$ts(j_2) \leq ts(j_1) + (1 - \beta(j_1,j_2,m)) \cdot \Lambda \tag{25}$$
$$1 \geq \beta(j_1,j_2,m) + \delta(j_1,m) \tag{26}$$
$$tf(j_2) - \epsilon - (1 - \eta(j_2,j_1)) \cdot \Lambda - (1 - \delta(j_2,m)) \cdot \Lambda \leq ts(j_1) + \beta(j_1,j_2,m) \cdot \Lambda \tag{27}$$

The above MILP formulation finds an assignment and schedule that minimize the chip peak temperature. We would like to point out that our formulation can readily be modified to produce an assignment and schedule that minimize peak power. We simply substitute the objective function by $\min P_{max}$ where

$$P_{max} \geq \forall_{i \in J} \sum_{m \in M} \sum_{j \in J} \beta(i,j,m) \cdot P(j,m). \tag{28}$$

On the other hand, if total energy is to be minimized, the following objective function can be used

$$E_{total} \geq \sum_{j \in J} \sum_{m \in M} P(j,m) \cdot E(j,m) \cdot \delta(j,m), \tag{29}$$

where E_{total} denotes the total energy. We will show in Section VII that directly minimizing peak temperature generally yields better results than approximating it with peak power or total energy minimization.

IV.B. Modeling Power Consumption

In Eq. 14, the parameter $P(j,m)$ captures the power consumption of core m while executing task j, and

$$P(j,m) = P_{dyn}(j,m) + P_{leak}(j,m) \tag{30}$$

where P_{dyn} and P_{leak} are the dynamic (switching) power and the leakage power due to executing task j on core m, respectively.

Assuming average switching activity is used to evaluate $P_{dyn}(j,m)$, we can treat $P_{dyn}(j,m)$ as a constant. The leakage power, $P_{leak}(j,m)$, however, is a superlinear function of temperature. Simply treating $P_{leak}(j,m)$ as a constant may lead to an underestimation of the chip peak temperature, causing hardware throttling at run-time, which may, in turns, cause hard real-time deadlines to be missed.

Though integrated circuit (IC) leakage power is a superlinear function of temperature, it is possible to approximate in the operating temperature ranges of integrated circuits using a piecewise linear function with only about 5% error in leakage estimation [40]. Therefore, we can model the power consumption required to execute a task j on core m at temperature T_m as

$$P(j,m) = K_1(j,m) \cdot T_m + K_2(j,m), \tag{31}$$

where $K_1(j,m)$ and $K_2(j,m)$ are constants that depend on core m and task j. Consequently, Eq. 14 can be rewritten as

$$T(i,m) \equiv T_h(i,h) + \frac{1}{G_h(m)} \sum_{n \in N_m} G_n(m,n) \cdot [T(i,n) - T(i,m)] + \frac{1}{G_h(m)} \cdot \left[\sum_{j \in J} \beta(i,j,m) \cdot (K_1(j,m) \cdot T(i,m) + K_2(j,m)) \right]. \tag{32}$$

To eliminate the nonlinear term $\beta(i,j,m) \cdot T(i,m)$, we replace $\beta(i,j,m) \cdot T(i,m)$ with a new variable $\lambda(i,j,m)$. In other words,

$$\lambda(i,j,m) = \begin{cases} K_1(j,m) \cdot T(i,m) & \text{if } \beta(i,j,m) = 1 \\ 0 & \text{otherwise,} \end{cases} \tag{33}$$

We then add the following constraints to our MILP formulation.

$$\forall m \in M, \forall i, j \in J$$

$$\lambda(i,j,m) \geq 0 \tag{34}$$

$$\lambda(i,j,m) \leq \beta(i,j,m) \cdot \Lambda \tag{35}$$

$$\lambda(i,j,m) \geq (K_1(j,m) \cdot T(i,m)) - (1 - \beta(i,j,m)) \cdot \Lambda \tag{36}$$

$$\lambda(i,j,m) \leq (K_1(j,m) \cdot T(i,m)) - (\beta(i,j,m) - 1) \cdot \Lambda \tag{37}$$

It can readily be verified that if $\lambda(i,j,m)$ satisfies Eq. 34–37, then Eq. 33 holds, which is precisely what the term $\beta(i,j,m) \cdot T(i,m)$ represents.

Therefore, solving the MILP instance given in Eq. 14, Eq. 15 (with $\beta(i,j,m) \cdot T(i,m)$ being replaced by $\lambda(i,j,m)$), Eq. 15–27, and Eq. 34–37 leads to an exact solution to the problem defined in Section II-C, assuming that steady-state analysis is acceptable.

IV.C. Incorporating Dynamic Voltage Scaling

Many modern processors support dynamic voltage and frequency scaling (DVFS). Although using DVFS to minimize energy will generally also reduce peak temperature, energy minimization alone is not equivalent to peak temperature minimization. Energy minimization does not consider temporal or spatial thermal variation. It is, however, possible and beneficial to consider DVFS in conjunction with our peak temperature optimization technique. Our MILP formulation from Section IV-A can be modified as follows.

For each core m, the set of discrete voltage levels, K_m, must be specified. We also redefine $E(j,k,m)$ to be the execution time of task j on core m at voltage level k and $P(j,k,m)$ to be the power consumption required to execute task j on core m at voltage level k. The binary variables $\delta(j,k,m)$ are also redefined to be 1 if task j is assigned to core m at voltage level k. Consequently, from Eq. 14,

$$\sum_{j \in J} \beta(i,j,m) \cdot P(j,m) = \sum_{k \in K} \sum_{j \in J} \nu(i,j,k,m) \cdot P(j,k,m),$$

where

$$\nu(j_1,j_2,k,m) = \begin{cases} 1 & \text{if } \delta(j_2,k,m) = 1,\ \beta(j_1,j_2,m) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

The constraints in Eq. 14–27 can be readily modified for use in the new formulation.

IV.D. Finer-Grained Thermal Model

The thermal model (Section II-B) can further be refined by using multiple thermal elements for each core, where each thermal element may have different power consumption and/or correspond to a particular functional unit of the core. We omit this refinement due to lack of realistic benchmarks for which power profile variations within cores are known. When such benchmarks become available, only minor modifications to the solution in Section IV-A will be needed. Specifically, the binary variables $\delta(j,m,x)$ must be redefined to take the value of 1 if j executes on core m and x is a thermal element belonging to core m. In addition, the variables in Eq. 8 must be modified accordingly.

It is worth noting that while a core power consumption may vary depending on the actual instructions that are being executed, the relative change in temperature at the functional-unit level is rather slow due to the relatively large RC time constant of the functional units. For this reason, the finer-grained thermal model presented here would be accurate enough and it is not necessary to model variation in instruction-level power consumption [41].

IV.E. Modeling Inter-Task Communication

In some situations, communication cost for a task to send data to its successors is not negligible. Given that the time to send data from task i to task j using shared memory is expressed by parameter $C(i,j)$, our MILP formulation from Section IV-A can be modified to capture inter-task communication by simply substituting Eq. 18 with the following

$$\forall i, j \in J \qquad ts(j) \geq (tf(i) + C(i,j)) \cdot \Gamma_{i,j}. \tag{38}$$

IV.F. Limitations of MILP-Based Approach

While the solution provided by the MILP formulation in Section IV-A is optimal, there are two main limitations to the MILP-based approach: 1) the MILP formulation cannot be used to efficiently solve large problem instances, as the problem defined in Section II-C is $\mathcal{NP}$-hard, and 2) due to the use of steady-state analysis, the MILP formulation may overestimate the chip peak temperature when task execution times are short compared to the RC thermal time constant of the cores (i.e., the constant influencing the rate of temperature change in response to power consumption change). That is, steady-state analysis can be used to accurately predict the temperature when task execution times are long compared to the core RC time constants, though transient analysis should otherwise be used to permit more accurate temperature prediction, thereby allowing more aggressive scheduling of short tasks that do not cause temperatures to converge to steady-state values during execution.

V. Assignment and Scheduling Heuristic Framework

To trade off accuracy in temperature estimation for running time, we propose an assignment and scheduling heuristic framework where either steady-state or transient analysis can be used, depending on the characteristics of the tasks under consideration. Additionally, our heuristic framework can be used in conjunction with any thermal modeling tool.

Our framework uses a binary search based approach to minimize peak temperature under functionality and timing constraints. It takes as inputs upper and lower temperature bounds, as well as the maximum number of iterations, *maxIter*. It then uses the average of the upper and lower bounds on the peak temperatures as the target peak temperature to find an assignment and schedule. If an assignment and schedule is found while staying below the target temperature, the current

target temperature will be used as the upper temperature bound for the next iteration of the binary search. Otherwise, it will be used as the lower temperature bound.

We introduce the key part of our framework: `ThermalSched`, a list scheduling [42] algorithm, which is summarized in Algorithm 1. For a given task j, the earliest start time (*EST(j)*) and latest start time (*LST(j)*) are computed. The mobility of task j can then be calculated as the difference between *LST(j)* and *EST(j)*. A potential challenge in computing *EST(j)* and *LST(j)* is that the execution time of task j is unknown prior to the selection of a core. Our solution is to use the smallest execution time of task j as given by the fastest core when computing *EST(j)* and *LST(j)* to maximize the mobility of task j, for all $j \in J$.

The steps for task assignment and scheduling follow. Ready tasks are ordered in a non-decreasing order of mobility. A *ready task* is a task whose predecessors have finished executing. Given a ready task j, `ThermalSched` selects the fastest available core that allows the task to meet its deadline while keeping the peak temperature below the target temperature. The fastest available core is chosen to maximize the mobility of the successors of task j, thereby improving schedulability. If no core is fast enough to execute task j by its deadline, `ThermalSched` terminates. (We ignore Lines 24–25 and 35–36 in Algorithm 1, and the variables *currentDelay* and *dMaxIter* for now. Their use will be explained in the next section.) Our search-based scheduling approach permits the use of an efficient list scheduler without global knowledge of temperature variation.

Observe that Algorithm 1 does not provide any details regarding the actual computation of the thermal profile (Line 17). Since predicting highly accurate thermal profiles increases time complexity, we propose to use two techniques based on the observations made in Section IV-F to balance accuracy and time complexity.

Algorithm 1 ThermalSched($G(V, E)$, *targetTemp*, *dMaxIter*)

1: compute *EST(j)*, for all tasks // earliest start time
2: compute *LST(j)*, for all tasks // latest start time
3: compute *avgE* // average execution time over all tasks and cores
4: *mobility(j)* ← *LST(j)* - *EST(j)*, for all tasks
5: *currentTime* ← 0
6: *busy(m)* ← 0, for all cores
7: **while** there are unscheduled tasks **do**
8: *RT* ← ready tasks in non-decreasing order of mobility
9: **for** each $j \in RT$ **do**
10: *invalidCount* ← 0
11: *fastestCore* ← -1
12: *bestExeTime* ← ∞
13: **for** each $m \in M$ **do**
14: $\delta(j, m) \leftarrow 0$
15: *endTime* ← *E(j,m)* + *currentTime*
16: **if not** *busy(m)* **and** *endTime* $\leq D(j)$ **then**
17: compute projected thermal profile for $[currentTime, nextIdleTime]$ // *nextIdleTime* is the next earliest time when all cores become idle
18: *peakTemp* ← $\max_{m \in M} T(j, m)$
19: **if** *peakTemp* $\leq$ *targetTemp* **then**
20: **if** *E(j,m)* $<$ *bestExeTime* **then**
21: *fastestCore* ← *m*
22: *bestExeTime* ← *E(j,m)*
23: *currentDelay* ← 0
24: **else if** *currentTime* > 0 **then**
25: [*fastestCore*, *bestExeTime*, *currentDelay*] ← DelayInsertion($G(V, E)$, j, m, *currentTime*, *targetTemp*, *dMaxIter*, *avgE*)
26: **else if not** *busy(m)* **then**
27: *invalidCount* ← *invalidCount* + 1
28: **if** *invalidCount* = $|M|$ **then**
29: **return** INFEASIBLE
30: **else if** *fastestCore* $\neq -1$ **then**
31: $\delta(j, fastestCore) \leftarrow 1$ // assign j to *fastestCore*
32: *ts(j)* ← *currentTime* + *currentDelay*
33: *tf(j)* ← *ts(j)* + *E(j, fastestCore)*
34: *busy(fastestCore)* ← 1
35: **if** *currentDelay* > 0 **then**
36: **break** // allow no tasks to start executing between *currentTime* and *currentTime* + *currentDelay*
37: update *EST(j)*, for all unscheduled tasks
38: update *mobility(j)*, for all unscheduled tasks
39: *nextSchedPoint* ← $\min\{tf(j) : tf(j) > currentTime + currentDelay\}$
40: **for** each $m \in M$ **do**
41: **if** m becomes idle at *nextSchedPoint* **then**
42: *busy(m)* ← 0
43: *currentTime* ← *nextSchedPoint*
44: **return** FEASIBLE

V.A. Steady-State Analysis Based Heuristic

As explained in Section IV-F, if task execution times are long compared to the thermal time constants of the cores, steady-state analysis can usually predict the resultant chip temperature in a fairly accurate manner. In such a case, the use of steady-state analysis reduces the running time of the heuristic.

The steady-state thermal profile can be computed by expressing Eq. 2 and Eq. 3 for all the thermal elements as a system of linear equations of the form $A \cdot \mathbf{T} + B = 0$, and of size $|E| \times |E|$, where $|E|$ is the total number of thermal elements. Since the thermal conductance matrix A is fixed once a floorplan is given, the inverse of the matrix can be pre-computed once and the temperature matrix can be updated using a constant number of multiplications in each iteration. `ThermalSched` therefore has a time complexity in $O(|J|^2 \cdot |M|^3)$. The time complexity of the **steady-state thermal analysis based heuristic (SSAB)** is in $O(|J|^2 \cdot |M|^3 \cdot maxIter)$.

V.B. Transient Analysis Based Heuristic

If task execution times are short, it is desirable to use transient analysis to compute the projected thermal profile, as explained in Section IV-F. Essentially, any existing thermal analysis technique can be used in our task assignment and scheduling heuristic framework. To validate our **transient analysis based heuristic (TAB)**, we will use `HotSpot` [36] in our experiments. Due to the use of transient analysis to predict the temperatures, the transient analysis based heuristic has a time complexity of $O(|J|^2 \cdot |M| \cdot maxIter \cdot O(HotSpot))$, where $O(HotSpot)$ is the worst-case running time of `HotSpot`.

A potential drawback of our flexible heuristic framework is that we always try to schedule as many tasks as possible in a given scheduling point provided that the chip peak temperature remains within the target temperature bound. In some situations, it may be better to schedule fewer tasks at a time to allow the chip to cool down before executing more tasks, potentially reducing the chip peak temperature. Based on this observation, we now introduce the concept of delay insertion.

VI. Delay Insertion

As stated previously, Algorithm 1 always tries to schedule as many tasks as possible at every scheduling point to maximize the mobility of later tasks. One possible consequence of this greedy approach to task assignment and scheduling is that the chip peak temperature may be so close to the target temperature that no future ready task can execute without violating the target temperature bound, thus requiring a higher target temperature to find a feasible task assignment and schedule. To address this weakness in our heuristic framework, we introduce the concept of delay insertion. That is, when the chip peak temperature is at or near the target peak temperature, we delay the execution of the next ready task by introducing an idle interval before the task starts to allow the chip to cool down. This improves the probability of later tasks being scheduled without exceeding the target temperature bound.

Since the steady-state thermal analysis assumes fast temperature rises and falls, an idle time (or a delay inserted) between task executions has no effect on the resultant peak temperature. On the other hand, transient thermal analysis would capture the cooling effects of delay insertions. Hence, the concept of delay insertions applies to the TAB heuristic only.

To demonstrate the potential benefits of delay insertions, we use the following simple example. Consider a system with 5 identical real-time tasks running on the MPSoC shown in Figure 1(b) with the associated execution time and power consumption for each core. Again, to obtain a chip power density that is similar to the the one for the 65 nm processors [38], we multiplied the power consumption of each core by a factor of 10. Without inserting any idle times, our TAB heuristic finds a feasible assignment and schedule with a peak temperature of 51.60 °C. Now, assume that there exists an algorithm that can insert the appropriate delays after some task executions, then the peak temperature could be reduced to 49.88 °C.

In the above example, delay insertion only reduces the chip peak temperature by 1.72 °C because there are only 5 tasks in the system with relatively short execution times. In other words, executing these tasks on the example MPSoC does not significantly raise the chip peak temperature. If our tasks require 10× the original execution times, i.e., a mean of 30 ms, then delay insertions would reduce the chip peak temperature by 3.87 °C, from 66.22 °C to 62.35 °C.

We now explain the use of delay insertions in our heuristic framework in more detail. Whenever an attempt to schedule a task on a core fails because the target temperature bound is exceeded, `DelayInsertion` is called (Line 24–25 of Algorithm 1). If an idle time has successfully been inserted into the schedule, our heuristic will immediately move on to the next scheduling point, i.e., by setting *currentTime* to $\min\{tf(j) : tf(j) > currentTime + currentDelay\}$ and continue the assignment and scheduling process (Lines 35–36 and Line 39 of Algorithm 1). No other tasks that are yet to be scheduled are allowed to run during $[currentTime, currentTime + currentDelay)$. This not only simplifies the algorithm, but is also quite reasonable since it is unlikely that when an idle time has been inserted (i.e., when the current chip peak temperature is at or near the target temperature), another task can execute within the interval $[currentTime, currentTime + delay)$ without exceeding the target temperature itself.

Algorithm 2 DelayInsertion($G(V, E)$, j, m, *currentTime*, *targetTemp*, *dMaxIter*, *avgE*)

```
1:  upperDelay ← avgE
2:  lowerDelay ← 0
3:  delay ← (upperDelay + lowerDelay) / 2
4:  iter ← 0
5:  oldPeakT ← 0
6:  newPeakT ← 0
7:  while iter < dMaxIter do
8:    compute projected thermal profile for
      [currentTime, currentTime + delay] and
      [currentTime + delay, nextIdleTime] // nextIdleTime
      is the next earliest time when all cores become idle
9:    newPeakT ← max_{m∈M} T(j, m)
10:   if targetTemp > newPeak and |oldPeakT − newPeakT|
      < ϵ then
11:     if E(j,m) + delay < bestExeTime and currentTime +
        delay + E(j, m) ≤ D(j) then
12:       fastestCore ← m
13:       bestExeTime ← E(j,m)
14:       return [fastestCore, bestExeTime, delay]
15:     else
16:       return FAILURE
17:   else if targetTemp > newPeakT then
18:     upperDelay ← delay
19:   else
20:     lowerDelay ← delay
21:   oldPeakT ← newPeakT
22:   delay ← (upperDelay + lowerDelay) / 2
23:   iter ← iter + 1
```

Our delay insertion algorithm is shown in Algorithm 2. To find the appropriate idle time to insert without sacrificing the

schedulability of future tasks, we use a binary search based approach. In each iteration, Algorithm 2 attempts to schedule the current task onto the core currently under consideration such that the resultant peak temperature does not exceed the target peak temperature. Should this prove to be possible, Algorithm 2 will keep this scheduling and assignment if the current configuration minimizes the task finish time until now. Hence, if an assignment and schedule exists for the current task that does not involve delay insertions, that assignment and schedule will likely be selected (this design choice maximizes the mobility of future tasks). Algorithm 2 will halt when either the maximum number of iterations *dMaxIer* has been reached or when an appropriate idle time has been found. The appropriate idle time is found when the current assignment and schedule does not exceed the target temperature and the chip peak temperature has converged (Line 10 of Algorithm 2).

In our implementation, the search begins by setting the upper bound on the delay to the average execution time of all task and core combinations and the lower bound delay to 0. We decided to use the average task execution times as the upper bound for the following reasons: 1) if the upper bound is too large, `DelayInsertion` can be unfortunately slow, and 2) `DelayInsertion` is most likely invoked when the chip temperature is near the target temperature. Inserting an idle time similar to the average task execution time would allow, on average, the chip to cool down enough to allow most tasks to eventually be scheduled on the current core without sacrificing the efficiency of the heuristic.

VII. Experimental Results

We quantify the benefits of our proposed approach and assess the quality of our heuristic framework in this section.

VII.A. Experimental Setup

In our experiments, we used the Embedded System Synthesis Benchmarks Suite (E3S) [34]. E3S contains 17 processing elements (PEs). From the E3S benchmarks, each PE is associated with a size, power value, and task execution times. Out of these 17 PEs, 12 have less than 3 W power consumption. In our experiments, we used the following 11 cores: AMD K6-2 450, AMD K6-2E 400 Mhz/ACR, AMD K6-2E+ 500 Mhz/ACR, AMD K6-IIIE+ 550 Mhz/ACR, IBM PowerPC 405GP 266 Mhz, IBM PowerPC 750CX 500 MHz, IDT32334 100 MHz, IDT79RC32V334-150, IDT79RC64575 250 MHz, Motorola MPC555 40 MHz, and TI TMS320C6203 300 MHz. (Note that we did not use all 17 cores because for each floorplan, we attempted to use cores with similar sizes.) The E3S task sets follow the organization of the EEMBC benchmarks [34]. There are five benchmarks in total: Auto (24 tasks), Consumer (12 tasks), Networking (13 tasks), Office (5 tasks), and Telecom (30 tasks). Each benchmark represents an application, as its name indicates. Each sink task, which does not have any successors, has a hard real-time deadline.

For the E3S benchmarks, we experimented with a number of floorplans with 2×2, 2×3, and 3×3 core arrangements. Each benchmark has different floorplans, as specific tasks are required to run on specific cores among the 11 cores mentioned above. The specific configuration of each floorplan is provided Table I and the corresponding core names in Table II. The chips consist of heterogeneous cores. Since cores with different power consumptions tend to have different areas, the vertical and lateral thermal conductance between neighboring cores, and between cores and heatsink elements will vary and can be computed as described in Section II-B.

TABLE I
Floorplan configurations.

Benchmark	First Row	Second Row	Third Row
Auto-2x2-1	14, 7	1, 7	
Auto-2x2-2	1, 7	1, 7	
Auto-2x3	1, 7, 7	1, 7, 7	
Auto-3x3	14, 14, 14	1, 1, 1	7, 7, 7
Consumer-2x2-1	7, 7	11, 11	
Consumer-2x2-2	7, 7	9, 7	
Consumer-2x3	7, 7, 7	11, 11, 11	
Consumer-3x3	7, 7, 7	9, 9, 9	7, 7, 7
Networking-2x2-1	2, 3	4, 5	
Networking-2x2-2	5, 5	4, 4	
Networking-2x3	5, 4, 5	5, 4, 5	
Networking-3x3	5, 5, 5	4, 4, 4	5, 5, 5
Office-2x2-1	4, 3	4, 12	
Office-2x2-2	4, 3	4, 3	
Office-2x3	4, 8, 4	3, 8, 3	
Office-3x3	3, 8, 3	8, 12, 8	3, 8, 3
Telecom-2x2-1	14, 7	1, 7	
Telecom-2x2-2	1, 7	1, 7	
Telecom-2x3	1, 7, 7	1, 7, 7	
Telecom-3x3	14, 14, 14	1, 1, 1	7, 7, 7

TABLE II
Core names.

Index	Core
1	AMD ElanSC520-133 MHz
2	AMD K6-2 450
3	AMD K6-2E 400Mhz/ACR
4	AMD K6-2E+ 500Mhz/ACR
5	AMD K6-IIIE+ 550Mhz/ACR
6	Analog Devices 21065L - 60 MHz
7	IBM PowerPC 405GP - 266 Mhz
8	IBM PowerPC 750CX - 500 MHz
9	IDT32334-100 MHz
10	IDT79RC32364-100
11	IDT79RC32V334-150
12	IDT79RC64575-250MHz
13	Imsys Cjip 40 Mhz
14	Motorola MPC555 - 40MHz
15	NEC VR5432 - 167 MHz
16	ST20C2 50 Mhz
17	TI TMS320C6203-300MHz

We also used TGFF [43], which is a pseudo-random task graph generator, in our experiment to generate 10 additional benchmarks. For each benchmark, there are up to 5 task graphs and the total number of tasks ranges from 4 to 29 tasks (this is similar to the number of tasks in the E3S benchmarks). Each task has at most 3 predecessors and 2 successors. A 2×2 core arrangement was used, with an average core width and height of 5 mm and an average power consumption of 10 W.

VII.B. MILP Formulation Performance

In this set of experiments, we used `CPLEX` with `AMPL` to solve instances of the MILP formulation in Section IV for optimal peak temperature, energy, and peak power. Each E3S

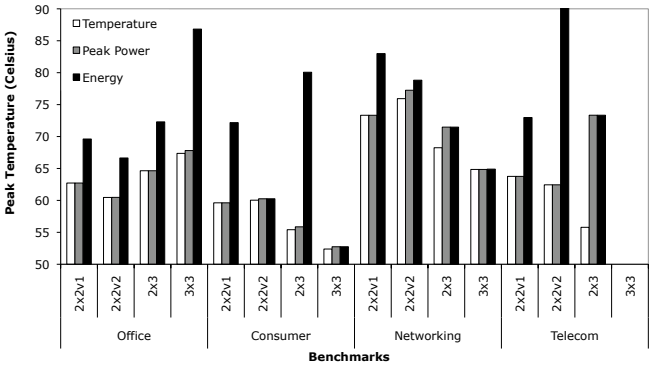


Fig. 3. Peak temp. minimization vs. energy and peak power minimization

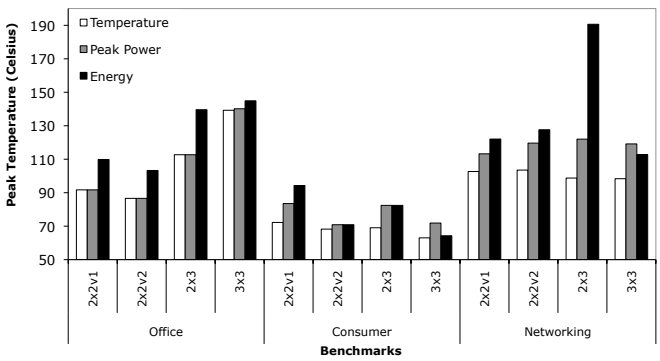


Fig. 4. Peak temp. minimization vs. energy and peak power minimization for higher power density chips

benchmark was run against two 2×2, one 2×3, and one 3×3 floorplan.

We first examine the temperature differences between using peak temperature minimization as the objective in the MILP formulation and using energy or peak power minimization. The solutions from the MILP solver are shown in Figure 3. The x-axis shows the different benchmarks and floorplans. The y-axis shows the resultant peak temperatures. Some results are unavailable due to the MILP solver running out of memory before finding a solution. Our approach reduces peak temperatures by up to 24.66 °C, and 9.19 °C on average, when compared to the method that minimizes energy. Most of the improvement results from considering the effects of temporal thermal variations.

The results in Figure 3 do not show significant differences in peak temperatures between our approach and the approach that minimizes peak power. This is because the low-power embedded cores used in our benchmarks have low power densities. For example, the floorplans for the Consumer benchmarks resulted in a chip power density ranging from 0.27 W/mm^2 to 0.36 W/mm^2 with an average chip power density of 0.32 W/mm^2. As a result, little spatial temperature variation was observed. However, spatial temperature variation will increase when higher power density chips are used, as explained in Section III.

To obtain chip power densities similar to those described by Link and Vijaykrishnan [38] for 65 nm processors, we multiplied each core's power consumption by 10. The resultant chip power densities ranged from 0.75 W/mm^2 to 3.13 W/mm^2 with an average power density of 2.28 W/mm^2. As shown in Figure 4, for these cores our method reduces peak temperatures by up to 23.25 °C, and 9.58 °C on average when compared to the method of peak power minimization and these results prove that spatial thermal variations need to be considered.

There exist situations where optimal peak temperature cannot be obtained by minimizing either energy or peak power. This situation was observed in the Networking benchmark with a 2×3 core arrangement (depicted in Figure 1(c), diagram not drawn to scale). Due to the characteristics of this benchmark, at least two cores must be active simultaneously at some point in time. In the case of energy and peak power minimization, the optimal solution consists of executing on cores *m2* and *m5* in parallel. This yields an optimal peak power of 28 W and an optimal energy of 35.46 J. The peak temperature obtained in this case is 61.45 °C. Using our approach, cores $m3$ and $m4$ execute simultaneously. This solution gives an optimal peak temperature of 58.15 °C, which is about 3 °C lower than either energy or peak power minimization. However, this solution yields a peak power of 32 W and a total energy of 36.20 J, which means that neither energy nor peak power minimization can achieve this optimal peak temperature.

Even in the cases where peak power (or energy) minimization can yield optimal peak temperature, it is still relevant to minimize peak temperature directly. First, there is no guarantee that an optimal peak temperature will be obtained by minimizing peak power, as the latter considers temporal thermal variation but ignores spatial thermal variation. Second, there may exist a range of possible peak temperatures as a result of a single optimal peak power. If such a range is large, the actual peak temperature of a chip can vary significantly.

To illustrate this scenario, we performed an additional experiment using a slightly modified Consumer benchmark. In this version, task deadlines were modified in such a way that at least two tasks must execute in parallel. We used a 2×3 floorplan with homogeneous processors. The experiment was run twice. In the first run, we used the original chip power density. The chip power density was then increased by a factor of 10 (once again to obtain similar power densities as described by Link and Vijaykrishnan [38]) in the second run. For this benchmark, there exist four distinct parallel core assignments yielding the same optimal peak power but different peak temperatures. The left bars in Figure 5 show the peak temperatures for each of these four assignments in the first run. The right bars show the range of possible peak temperatures of the chip with a higher power density. The results show that the difference in peak temperature for the same peak power can be over 5 °C for the higher power density chip. Clearly, peak power minimization is not sufficient, especially when it is predicted that the power density of future chips will continue to increase, resulting in even higher spatial thermal variation.

Finally, when using the TGFF benchmarks described earlier, the MILP solved eight out of ten problem instances. Minimiz-

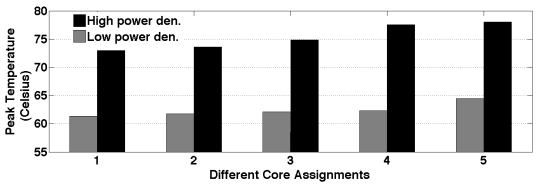


Fig. 5. Bar plot showing a range of peak temperatures using peak power minimization

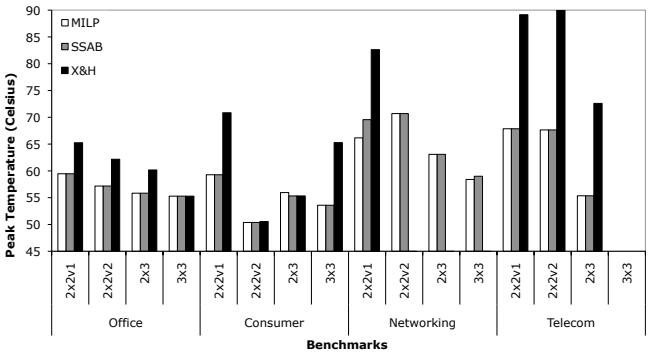


Fig. 6. Perf. of steady-state analysis based heuristics (based on HotSpot)

ing peak temperature directly instead of minimizing energy reduced peak temperatures by 9.41 °C on average and up to 24.19 °C. In addition, when compared to the method of peak power minimization, peak temperature minimization reduced peak temperatures by 1.27 °C on average and up to 6.71 °C.

The above results allow for a general conclusion to be drawn. Average power minimization ignores both temporal and spatial thermal variation, while peak power minimization only considers temporal thermal variation. Peak temperature minimization takes both types of thermal variation into account. In addition, task mobility given a floorplan and other tasks in the same benchmark explain why our approach obtains a higher peak temperature reduction for some benchmarks than the others.

VII.C. Performance of Steady-State Analysis Based Heuristic

We assess the performance of our SSAB algorithm (Section V) by comparing its solutions to the ones from the MILP (Section IV-A) as well as the results from Xie's and Hung's *Heuristic 1* [11], which we refer to as the X&H heuristic. The X&H heuristic calls HotSpot to compute the temperatures. Figure 6 compares the results from the SSAB and X&H heuristics to the optimal solution from the MILP formulation. We used HotSpot to compare the peak temperatures for a fair comparison. Results for benchmarks that were not successfully solved by the X&H algorithm are omitted.

The X&H heuristic deviates from the optimal solution by 10.94 °C on average and 38.40 °C in the worst case. On the other hand, the SSAB heuristic finds an optimal solution in many cases while giving results that deviate by at most 3.40 °C from optimality (and 0.22 °C on average) requiring at most 50 binary search iterations for each benchmark. Both heuristics require similar running times, but the SSAB heuristic never performs worse than the X&H heuristic.

When the X&H heuristic was tested against the 10 TGFF benchmarks mentioned previously, we found that it could only solve two of the benchmarks, with a maximum deviation from optimality of 7.74 °C. On the other hand, the SSAB heuristic could solve four, one of which was a benchmark that the MILP solver could not find a solution to. For the three other benchmarks, the SSAB heuristic found the same solutions as the MILP solver.

To demonstrate that the SSAB heuristic can efficiently solve larger problem instances, we considered a benchmark that consists of 30 tasks and a 4×4 core arrangement of homogeneous processors. First, we attempted to use the MILP solver. As expected, no solution was returned, as the 3.58 GB RAM workstation on which `CPLEX` was executing ran out of memory. On the other hand, the SSAB heuristic was able to find a solution using no more than 50 binary search iterations within 9 seconds.

On a final note, our thermal model is a discretized Fourier heat flow model and while it is not exactly the same model used in HotSpot, the experiments in this section showed that on average, the peak temperatures from the two models differed by less than $< 5\,^{\circ}\mathrm{C}$. This indirectly served as a validation of our thermal model.

VII.D. Performance of Transient Analysis Based Heuristic

We now assess the performance of the TAB heuristic (Section V) using the same set of benchmarks as in Section VII-A. The TAB heuristic calls HotSpot to determine transient temperatures. Since the original task execution times for the E3S benchmarks tend to be short, dynamic thermal effects can be significant. We compare the peak temperatures obtained by the MILP solver and the TAB heuristic, as shown in Figure 7. When compared to the results from the MILP solver, the TAB heuristic reduces the peak temperature by up to 0.67 °C and 0.06 °C on average. This is because transient analysis can more accurately predict temperatures when performing assignment and scheduling.

The TAB heuristic also improves the task finish times. Let the speedup be the ratio of the finish time of the last task in the MILP schedule to that in the TAB schedule. The maximum, minimum, and average speedups are 78.13×, 1.21×, and 9.02×, respectively. Such a significant speedup results from the TAB heuristic being much less pessimistic in estimating temperatures and hence scheduling more tasks in parallel. However, the SSAB heuristic is more efficient than the TAB heuristic. Specifically, the SSAB heuristic is about 175× faster than the TAB heuristic on average for benchmarks with short task execution times; this difference further increases for benchmarks with longer task execution times.

VII.E. Performance of Transient Analysis Based Heuristic with Delay Insertions

To determine the impacts of delay insertions (Section VI) on reducing the chip peak temperature, we once again used

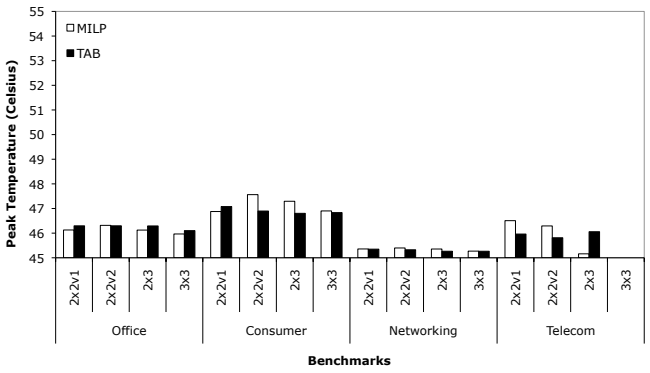


Fig. 7. Perf. of transient analysis based heuristic (based on HotSpot)

the E3S and TGFF benchmarks. We compared the solutions from the original TAB heuristic to those from the improved TAB heuristic (iTAB). Once again, since the power density of the E3S cores are quite low compared to the values described by Link and Vijaykrishnan [38], we multiplied each core's power consumption by a factor of 10. The results for the E3S benchmarks are shown in Table III. The first column shows the benchmark names and associated floorplans. The second main column presents the peak temperatures from the TAB heuristic and the iTAB heuristic, as well as the differences in peak temperatures for the original task execution times. Finally, the last main column presents the data for the case where task execution times are multiplied by a factor of 10.

With the original task execution times, the effect of delay insertions is minimal for some benchmarks (e.g., networking). In fact, for two of the telecom benchmarks, iTAB actually performs worse than TAB. This is because iTAB always tries to insert delays, even when it may not be optimal to do so. For instance, if the chip is currently too hot to execute the next ready task within the target peak temperature, iTAB would try to insert some idle time before scheduling that task. However, it may sometimes be better to select a different task that can meet the target peak temperature constraint in Algorithm 1 without inserting delays.

When the execution times are increased by a factor of 10, we see the benefits of using the iTAB heuristic. This is because the average chip peak temperature is much higher than in the original cases and inserting idle times between task executions is very effective in cooling the chip down. On average, iTAB produces solutions that reduce peak temperatures by 3.15 °C on average and up to 11.92 °C.

iTAB did not significantly improve on the TGFF benchmark solutions found by TAB (both found four out of ten). The largest improvement in peak temperature was 1.71 °C. This is because most tasks in the TGFF benchmarks did not have enough slack for delay insertions to be effective.

Based on the above results, we can conclude that iTAB reduces the peak temperature of systems with time slack. It must be noted, however, that iTAB also requires longer running times. On average, iTAB takes 19.2× longer to run than the original TAB algorithm. Clearly, there is a trade-off between solution quality and time complexity of these algorithms.

TABLE III
EFFECTIVENESS OF DELAY INSERTIONS IN REDUCING CHIP PEAK TEMPERATURES

Benchmark		Temperature (°C)					
		Original Execution Times			Execution Times × 10		
		TAB	iTAB	Diff.	TAB	iTAB	Diff.
Consumer	2×2-1	65.78	60.56	5.22	84.30	80.90	3.40
	2×2-2	86.24	81.47	4.77	86.24	81.47	4.77
	2×3	63.32	60.06	3.26	79.24	76.81	2.43
	3×3	61.97	60.28	1.69	78.60	76.85	1.75
Networking	2×2-1	47.49	47.45	0.04	57.81	55.43	2.38
	2×2-2	47.29	46.85	0.45	57.83	53.48	4.35
	2×3	46.80	46.80	0.00	53.96	53.87	0.09
	3×3	46.80	46.79	0.01	53.45	53.36	0.09
Office	2×2-1	54.22	54.22	0.00	75.96	75.96	0.00
	2×2-2	54.14	54.14	0.00	75.37	75.37	0.00
	2×3	54.21	54.21	0.00	67.91	67.89	0.02
	3×3	54.13	54.13	0.00	67.43	57.94	9.49
Telecom	2×2-1	50.28	51.70	-1.42	72.06	65.53	6.53
	2×2-2	49.48	52.79	-3.31	71.26	59.34	11.92
	2×3	46.38	47.40	-1.02	51.37	51.30	0.08

VIII. CONCLUSIONS AND FUTURE WORK

We presented a formal assignment and scheduling technique that uses a mixed-integer linear program solver to optimize IC peak temperature under precedence and hard real-time constraints based on phased steady-state thermal analysis. Experimental results showed a peak temperature reduction of up to 30.75 °C and 10.09 °C on average for embedded processors when compared to energy minimization. When compared to peak power minimization, our approach reduced peak temperature by up to 23.25 °C and 8.98 °C on average for high power density chips.

To efficiently solve this $\mathcal{NP}$-hard assignment and scheduling problem, we also proposed a task assignment and scheduling heuristic framework in which the actual method for temperature prediction depends on task durations. Phased steady-state analysis is appropriate when task execution times are long compared to the time constants of the cores and transient analysis should be used otherwise. Our phased steady-state analysis based heuristic finds an optimal solution in many cases, with a maximum deviation from optimality of 3.40 °C. When compared to previous work, the heuristic achieves a temperature reduction of 10.94 °C on average. The transient analysis based heuristic models and exploits the transient thermal effects of short tasks to further improve upon the existing solution by 0.67 °C in the best case. Finally, we showed that incorporating the concept of delay insertion into the proposed heuristic framework results in an additional peak temperature reduction of up to 11.92 °C.

Since real-time systems can exhibit great temperature variations at run-time due to the differences in actual task execution times, we intend on exploring the peak temperature minimization problem online to further reduce temperature and increase system reliability.

REFERENCES

[1] E. Milchman, "Intel dual-core FAQ," *Wired News*, July 2006.

[2] S. Y. Borkar, et al., "Platform 2015: Intel processor and platform evolution for the next decade," Intel Corporation, Tech. Rep., Mar. 2005.

[3] R. Kumar, D. M. Tullsen, and N. P. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in *Proc. Int. Conf. Parallel Architectures and Compilation Techniques*, Sept. 2006, pp. 23–32.

[4] "Xbox360 Xenon," 2006, http://domino.research.ibm.com/comm/research_projects.nsf/pages/multicore.Xbox360.html.

[5] "P4040: QorIQ Embedded Multicore Processor," 2009, http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=P4040&fsrch=1.

[6] J. Srinivasan, et al., "Exploiting structural duplication for lifetime reliability enhancement," in *Proc. Int. Symp. Computer Architecture*, June 2005, pp. 520–531.

[7] R. Viswanath, et al., "Thermal performance challenges from silicon to systems," *Intel Technology Journal*, vol. 4, no. 3, pp. 1–16, Aug. 2000.

[8] S. H. Gunther, et al., "Managing the impact of increasing microprocessor power consumption," *Intel Technology Journal*, vol. 5, no. 1, pp. 1–9, Feb. 2001.

[9] Y. Li, et al., "Performance, energy, and thermal considerations for SMT and CMP architectures," in *Proc. Int. Symp. Computer Architecture*, Feb. 2005, pp. 71–82.

[10] T. Zhou, X. Hu, and E.-M. Sha, "Probabilistic performance estimation for real-time embedded systems," in *Int. Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, Mar. 1999, pp. 83–88.

[11] Y. Xie and W.-L. Hung, "Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (MPSoC) design," *J. VLSI Signal Processing*, vol. 45, no. 3, pp. 177–189, Dec. 2006.

[12] J.-J. Chen, S. Wang, and L. Thiele, "Proactive speed scheduling for real-time tasks under thermal constraints," in *Proc. Real-Time and Embedded Technology and Applications Symp.*, Apr. 2009, pp. 141–150.

[13] S. Wang and R. Bettati, "Delay analysis in temperature-constrained hard real-time systems with general task arrivals," in *Proc. Real-Time Systems Symp.*, Dec. 2006, pp. 323–332.

[14] J. W. S. Liu, *Real-Time Systems*. Prentice-Hall, NJ, 2000.

[15] R. Mukherjee, S. Öğrenci Memik, and G. Memik, "Temperature-aware resource allocation and binding in high-level synthesis," in *Proc. Design Automation Conf.*, June 2005, pp. 196–201.

[16] P. Lim and T. Kim, "Thermal-aware high-level synthesis based on network flow method," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2006, pp. 124–129.

[17] R. Rao, et al., "An optimal analytical solution for processor speed control with thermal constraints," in *Proc. Int. Symp. Low Power Electronics & Design*, Oct. 2006, pp. 292–297.

[18] R. Rao and S. Vrudhula, "Performance optimal processor throttling under thermal constraints," in *Int. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, Oct. 2007, pp. 257–266.

[19] A. Mutapcic, et al., "Processor speed control with thermal constraints," *IEEE Trans. Circuits and Systems I*, 2009, to appear.

[20] G. Quan, et al., "Guaranteed scheduling for repetitive hard real-time tasks under the maximum temperature constraints," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2008, pp. 267–272.

[21] N. Bansal, T. Kimbrel, and K. Pruhs, "Dynamic speed scaling to manage energy and temperature," in *Proc. Symposium on Foundations of Computer Science*, Oct. 2004, pp. 520–529.

[22] R. Jayaseelan and T. Mitra, "Temperature aware task sequencing and voltage scaling," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2008, pp. 618–623.

[23] P. Gai, L. Abeni, and G. Buttazzo, "Multiprocessor DSP scheduling in system-on-a-chip architectures," in *Proc. Euromicro Conf. Real-Time Systems*, June 2002, pp. 231–238.

[24] E. Seo, Y. Koo, and J. Lee, "Dynamic repartitioning of real-time schedule on a multicore processor for energy efficiency," in *Proc. Int. Conf. Embedded and Ubiquitous Computing*, Aug. 2006, pp. 69–78.

[25] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Trans. Parallel & Distributed Systems*, vol. 4, no. 2, pp. 175–187, Feb. 1993.

[26] R. Rao and S. Vrudhula, "Efficient online computation of core speeds to maximize the throughput of thermally constrained multi-core processors," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2008, pp. 537–542.

[27] F. Mulas, et al., "Thermal balancing policy for streaming computing on multiprocessor architectures," in *Proc. Design, Automation & Test in Europe Conf.*, Mar. 2008, pp. 734–739.

[28] A. K. Coskun, T. S. Rosing, and K. Gross, "Temperature management in multiprocessor SoCs using online learning," in *Proc. Design Automation Conf.*, June 2008, pp. 890–893.

[29] A. K. Coskun, et al., "Temperature-aware MPSoC scheduling for reducing hot spots and gradients," in *Proc. Asia & South Pacific Design Automation Conf.*, Jan. 2008, pp. 49–54.

[30] H. Jung, P. Rong, and M. Pedram, "Stochastic modeling of a thermally-managed multi-core system," in *Proc. Design Automation Conf.*, June 2008, pp. 728–733.

[31] J.-J. Chen, C.-M. Hung, and T.-W. Kuo, "On the minimization of the instantaneous temperature for periodic real-time tasks," in *Proc. Real-Time and Embedded Technology and Applications Symp.*, Apr. 2007, pp. 236–248.

[32] C. Sun, L. Shang, and R. P. Dick, "Three-dimensional multi-processor system-on-chip thermal optimization," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2007, pp. 117–122.

[33] G. Paci, et al., "Exploring "temperature-aware design" in low-power MPSoCs," in *Proc. Design, Automation & Test in Europe Conf.*, Mar. 2006, pp. 838–843.

[34] "Embedded microprocessor benchmark consortium," http://www.eembc.org.

[35] E. L. Lawler and C. U. Martel, "Scheduling periodically occurring tasks on multiple processors," *Information Processing Ltrs.*, vol. 7, pp. 9–12, Feb. 1981.

[36] K. Skadron, et al., "Temperature-aware microarchitecture," in *Proc. Int. Symp. Computer Architecture*, June 2003, pp. 2–13.

[37] R. A. Serway, *Physics for Scientists & Engineers with Modern Physics*. Saunders College Publishing, 1990.

[38] G. Link and N. Vijaykrishnan, "Thermal trends in emerging technologies," in *Proc. Int. Symp. Quality of Electronic Design*, Mar. 2006, pp. 625–632.

[39] L.-F. Leung, C.-Y. Tsui, and W.-H. Ki, "Simultaneous task allocation, scheduling and voltage assignment for multiple-processors-core systems using mixed integer nonlinear programming," in *Prof. Int. Symp. Circuits and Systems*, May 2003, pp. 309–312.

[40] Y. Liu, et al., "Accurate temperature-dependent integrated circuit leakage power estimation is easy," in *Proc. Design, Automation & Test in Europe Conf.*, Mar. 2007, pp. 204–209.

[41] N. Allec, et al., "ThermalScope: multi-scale thermal analysis for nanometer-scale integrated circuits," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2008.

[42] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Book Company, NY, 1994.

[43] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Proc. Int. Wkshp. Hardware/Software Co-Design*, Mar. 1998, pp. 97–101.