Thermal-Aware Management Techniques for Cyber-Physical Systems

C.M. Krishna and I. Koren
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst

***Abstract*—The power density of processors has increased greatly over time. Since elevated temperatures greatly shorten the lifetime of semiconductor devices, thermal management has emerged as a key topic in the design and control of computational platforms. In this paper, we provide a comprehensive yet compact survey of thermal management in cyber-physical systems. Such systems are constrained by the need to meet hard deadlines; this distinguishes them from general-purpose systems and motivates distinctive resource-management approaches.**

I. Introduction

Elevated temperatures rapidly accelerate chip death. With average chip temperature rising, heating-related failures are a serious concern. Just a few degrees' rise in temperature can halve the mean lifetime of a chip [1]. At the same time, high-speed processing that results in inceased temperatures is essential for many cyber-physical applications. The question of how to take thermal considerations into view while scheduling the real-time workload to meet all hard deadlines is therefore of considerable practical importance. The purpose of real-time scheduling is to meet all task deadlines, especially those of safety-critical tasks. A significant literature on the subject has grown over the past four decades. Introducing thermal constraints expands the problem into another dimension, requiring modification of traditional resource management algorithms.

The purpose of this paper is to provide an overview of the resource-management approaches currently taken in this field. While this paper is largely self-contained in its treatment of the major topics in thermal-aware scheduling, it largely complements two previously published surveys [2], [3]. We focus here on the particular requirements of real-time cyber-physical systems while [2], [3] dealt more broadly with applications as well as aspects of chip design and heat flow modeling.

This paper is organized as follows. In Section II, we provide some technical background; both the impact of high temperature on solid-state devices and the nature of cyber-physical systems are discussed. We then turn in Section III to the optimization criteria used in thermal-aware scheduling and discuss how well they capture the underlying vulnerability of devices to heating and how easy they are to compute. In Section IV, we review ways to measure or estimate the on-chip temperature. Since most thermal scheduling algorithms work by considering the current on-chip temperature, it is important that rapid, low-overhead, and accurate, temperature measurement/estimation techniques be available. Major challenges in doing so arise from the process variation that exists from chip to chip and also the wide variation in the thermal impact of processes from one execution to the next (due to the impact of input data on their execution path). In Section V, we present the heat flow equation that is at the heart of almost all contemporary thermal scheduling research. This is a linear equation and the convenience of linearity is explored at some depth. Also discussed is the issue of model granularity: how fine-grained is the heat flow model used in practice? Thermal control options are described in Section VI; almost all thermal scheduling approaches consist of using one or more of these. Given these options, Section VII covers *reactive* and *proactive* ways of deploying them. Real-time thermal-aware scheduling issues are covered in Section VIII and the paper concludes with a discussion in Section IX.

II. Technical Background

A. Cyber-Physical Workloads

A cyber-physical system (CPS) consists of two major parts: (1) the cyber component, consisting of the controller which runs the control algorithms to compute control inputs, and (2) the physical component, in the form of a physical plant, which is being controlled. A very wide variety of applications are covered under this category: the physical plant may range in size and scope from an implanted medical device to a system for controlling a continent-wide power grid.

The cyber component – the controller – is in the feedback loop of the controlled plant. There are two aspects of its activity which affect the quality of the control it affords. One is the response time of its tasks; the other is the quality of the control algorithm it executes.

The response time affects feedback delay. We know from basic control theory (and, indeed, from common sense) that the greater the feedback delay, the worse the quality of control tends to be. In fact, beyond a certain delay, the controlled plant can actually become unstable. This response time is a function of two things: the capability of the computational platform and the intensity of its workload. Typically, control tasks are run either periodically or sporadically. Periodic tasks are released at regular intervals; sporadic tasks are run whenever triggered by a human operator or the occurrence of some event in the operating environment, under the condition that they will not be invoked more often than a specified number of times per unit time. Standard techniques from digital control theory are used to determine the appropriate rate at which periodic tasks must be dispatched.

The quality of the control algorithm can affect its complexity and its robustness. As its complexity increases, so does the intensity of the computational workload.

A key distinguishing feature of CPS applications from general-purpose ones is that the CPS workload is usually well characterized in advance. Whether the application is a fly-by-wire aircraft or an implanted medical device, the set of control tasks required is well known in advance. Each of these tasks can be analyzed and profiled ahead of actual use in the CPS. As a result, we have much more detailed information as to the worst-case computational demand imposed by each of the computational tasks in a CPS. Hence, decisions on provisioning the cyber side to be able to meet all control task deadlines can be made with confidence.

B. Thermally Accelerated Failure

There are several causes of thermally accelerated failures and often, the acceleration is exponential in the temperature. The expressions quoted here are from the well-known RAMP model [4]. Several of them (listed below) follow the so-called Arrhenius model, where the failure rate is exponentially dependent on an activation energy, E_a, divided by the absolute temperature, T.

Electromigration, as the term implies, is the physical movement of metal atoms due to heating; this can cause open and short circuits [5]. The Mean Time to Failure (MTTF) due to this failure mode is

$$MTTF_{EM} = K_{EM} J^{-n} e^{\frac{E_{a-EM}}{kT}} \tag{1}$$

where E_{a-EM} is the activation energy (typically 0.9 eV for Copper) and J is the current density. n is about 1.1 for Copper, k is Boltzmann's constant and K_{EM} is a constant of proportionality.

Stress migration is a failure due to unequal expansion of different materials under heating. The mean time to failure under this mode is

$$MTTF_{SM} = K_{SM} |T_0 - T|^{-m} e^{\frac{E_{a-SM}}{kT}} \tag{2}$$

where T_0 is the temperature at which metal deposition was made, E_{a-SM} is the activation energy (0.9 for Copper) and $m \approx 2.5$ for Copper; K_{SM} is a constant of proportionality.

Time-Dependent Dielectric Breakdown is a breakdown of the dielectric; as transistor gates become ever thinner, their chances of being punched through goes up. The MTTF for this failure mode has the expression

$$MTTF_{TDDB} = K_{TDDB} V^{-(a-bT)} e^{\frac{X+Y/T+ZT}{kT}} \tag{3}$$

where V is the supply voltage, and a, b, X, Y, Z are constants; K_{TDDB} is a constant of proportionality.

Thermal cycling is due to repeated heating and cooling of the chip. For slow cycling, the mean time to failure (due to this mode) is inversely proportional to a power of the temperature swing. No good models are quoted for rapid thermal cycling.

The effect of the different thermal failure modes can be combined by adding up the rates of each of them. The rate of failure is approximated by taking the reciprocal of the MTTF of each mode. In other words, the total rate of failure is approximated by $\lambda_{tot} = \sum_{i=1}^{m} \lambda_i$ where $\lambda_i = \frac{1}{MTTF_i}$ and the significant failure modes are numbered 1 to m. Reliability over a given interval is conventionally defined as the probability that the system suffers no failure over that interval. The reliability over an interval $[0, \tau]$, and at a fixed temperature T, is then approximated by $\mathcal{R}_T(\tau) \approx e^{-\lambda_{tot}\tau}$.

Note that the MTTF expressions assume a fixed temperature, T. However, cyber physical systems often operate in environments where the temperature varies. In such a case, we can take a piecewise approach [6], dividing the time axis into short enough segments so that the temperature can be approximated as constant over each segment. Let T_j denote the approximated constant temperature for segment j. The probability of failing in segment j can be approximated as $p_j = \mathcal{R}_{T_j}([j-1]\Delta) - \mathcal{R}_{T_j}(j\Delta)$. The probability of failing sometime in the first k segments is then approximated by $\sum_{j=1}^{k} p_j$; the reliability over the interval $[0, k\Delta]$ is thus approximately $1 - \sum_{i=j}^{k} p_j$.

III. Thermal Scheduling Criteria

The simplest – and perhaps the most widely used – thermal scheduling criterion of all is a peak temperature constraint. That is, the system tries to optimize some aspect of the schedule while ensuring that the maximum temperature of the chip does not exceed a user-specified bound. For example, we might have a scheme whereby the processor is power-gated when it reaches its peak temperature constraint (to keep it from getting any hotter) and stays off until it cools to a lower threshold. At this point, it starts up again. The problem in such a situation is to determine schedulability tests which ensure that a given real-time workload is completed by the time available, despite time lost to power gating.

The peak temperature constraint, however, does not account for the fact that thermal damage tends to be cumulative. For example, metal atoms that migrate away from a wire when heated do not magically migrate back into place once the device cools. The *Accumulated Thermal Impact* (ATI) at any point in time is defined as the integral of all the processor temperatures up to that point [7]. This can be related to the workload by calculating the ATI at an instant meaningful to the workload. For example, if given a periodic workload, the instant of interest might be the LCM (least common multiple) of the task periods or some multiple thereof. Related to this is the *thermal utilization* measure of a periodic task, which is defined as its ATI multiplied by its dispatch frequency (thereby counting the rate at which that task generates ATI) and divided by the maximum temperature limit[1]. A thermal scheduling constraint can then be introduced so that the total thermal utilization of the entire task set will never exceed 1. The analogy with task utilization in traditional real-time scheduling [8] is obvious.

While ATI accounts for temperature history, it does not account for the nonlinear impact of temperature on the damage

[1] In reality, the work in [7] uses a *function* of the actual or physical temperature (rather than the actual temperature itself) in expressing the ATI and temperature limit, but that is a detail for algebraic convenience and has no conceptual implications.

inflicted on the chip. We have already seen (in Section II-B) that for many failure modes, the failure rates are exponentially accelerated by temperature. In addition, repeatedly heating and cooling can itself cause damage. Such failure acceleration can be directly taken into account during thermal management [9]. The *Thermal Age Acceleration Factor*, (TAAF), seeks to measure this quantity [10]. If $\eta(t)$ is the age acceleration induced at time t by heating, then $\int_0^\tau \eta(t)dt$ is the effective age of the device at time τ. TAAF is the ratio of the effective age to the chronological age. This does require a detailed model of failure rate as a function of temperature; such models applicable to the latest devices are not easy to find. Furthermore, the above expression ignores the impact of cycles of heating and cooling and it is not clear how one can capture this impact effectively since it depends on the rate of heating or cooling as well as the range of temperatures reached. One might speculate that such a model might be created based on the Fourier transform of the temperature curve (capturing the rate and amplitude of temperature changes); however, we are not aware of any experimentally well-validated model of thermal cycling for modern deep-submicron devices. It is up to the user to decide whether the additional information captured by the thermal age acceleration factor is worth more than the inaccuracies resulting from a simplified model of failure acceleration. As things currently stand, TAAF is useful when there is one dominant thermally-accelerated driver and when thermal cycling between hot and cold does not play a large role.

Thus far, we have presented criteria which solely focus on the processor. However, an argument can be sometimes made for considering the impact of the processor on its surroundings. Indeed, in some cases, the concern is more about the damage to the surrounding environment (caused by elevated processor temperature) than to the processor. This is justified in the case of implantable medical devices (IMDs). IMDs are surrounded by living tissue which is highly sensitive to being heated up. We must, therefore, balance the need for an IMD to maintain an adequate level of computational activity to fulfil its function while limiting (or avoiding) any thermal damage to surrounding tissue. A measure called the *real-time thermal resiliency* (RTR) has been proposed [11]. This measure, which is along the lines of the traditional performability measure for real-time systems [12], identifies distinctive levels of functionality (or performance) on the part of the IMD. The thermal resiliency function, $RTR(M_i, T_{proc})$, is the external (ambient) temperature at which the processor can maintain its temperature at T_{proc} while delivering a level of functionality associated with performance mode M_i or higher. (For example, M_i could denote the successful execution of a given set of real-time tasks dispatched at a specified rate.)

Each of these measures is a tradeoff between accuracy (in capturing the extent of thermal damage) on the one hand and complexity on the other. The peak temperature criterion is effective when we want to prevent the temperature from rising to such a point that failure becomes imminent. It is simple to use; it is useful when the application does not have long intervals during which repair/replacement is not available. ATI is slightly more complicated, involving as it does the integration of temperatures over time; it is accurate when the chip temperatures are either always quite low or when they are usually in a fairly narrow range. In both of these cases, the thermal damage is reasonably accurately modeled as a linear function of the temperature. TAAF is the most complicated – and accurate – of these measures. It should be used in conditions where chip temperatures may vary widely with time and when accuracy of thermal damage estimates is important (for example in applications which must function unattended for long periods without being serviced). All of these measures focus on the damage done to the computer system as a result of elevated temperature. RTI must be used when such elevated processor temperatures can cause damage to the operating environment as well.

IV. Measuring/Estimating Temperature

There are several ways to estimate on-chip temperature. The most direct method is to embed thermal sensors on the chip. An indirect approach is to use performance counters (which count the number of certain events in specified portions of the chip) and then use a model to relate a vector of such counts to the temperature. The third – and most indirect – approach is to estimate the power consumption of the chip (either in the aggregate or for various parts of the chip) and use heat flow models. In this section, we will concentrate on the first two approaches; heat flow models are presented in Section V.

A. Thermal Sensors

On-chip thermal sensors exploit the dependence on temperature of (*a*) the forward resistance of a thermal diode, or (*b*) the signal propagation delay. The former is used in analog sensors; the latter in digital ones.

It is well known that the forward resistance of a diode is dependent on temperature [13]. Thus, a small voltage can be imposed across a diode and the resulting current measured. Based on this, the forward resistance can be calculated and the temperature looked up from a conversion/calibration table. Note that the current level needs to be kept low, since any current flow causes heating of its own and can affect the measurements. At the same time, the current flow has to be large enough to not be greatly affected by prevailing noise.

It is also known that signal propagation delay increases with temperature. This can be used to measure temperature by setting up a ring oscillator [14]. Such a circuit consists of an odd number of diodes, connected head-to-tail, in a ring formation. It is easy to show that such a circuit will start oscillating, with the oscillation frequency being related to the signal propagation delay. Thus, we can count the number of oscillations in a given time period and then use a conversion table to estimate the prevailing temperature.

A common problem that both types of thermal sensors encounter is process variation. No two transistors are exactly alike; even under the most tightly controlled manufacturing conditions, and as a result, circuits vary in their behavior. The trend towards ever-smaller feature sizes makes the process variation problem worse and can easily lead to significant measurement errors. For example, in an experiment, an actual

temperature of 35^oC was "measured" in the range of 13^o to 46^oC while an actual temperature of 95^oC was reported in the range 61^o to 109^oC [15], [16].

Process variation effects can be countered by calibrating individual sensors. This can be done, for example, by placing the chip in a temperature-controlled environment and collecting data for a calibration table. Recently, researchers have suggested designing circuits which have a compensating component making them somewhat immune to process variation [15].

B. Counter-Based Approaches

Modern processors have several built-in performance counters. These are counters which can be tasked with recording the number of specified events. Counter values can therefore be used to reflect the level of processor activity [17], [18]. Since power consumption – and therefore temperature – is linked to such activity, we can use performance counter readings to estimate on-chip temperature.

This approach relies on the strong correlation between the performance counter readings, the activity levels in the various units within the processor, and the power consumed by these units [19], [20], [21]. This allows us to calculate the total power consumption as a function of the performance counter readings. Experiments are run for a large number of benchmark workloads and the performance counters' values and power estimates collected. Heat flow models (see Section V) can then estimate the temperature from the power dissipation. Standard regression techniques are then used to obtain a formula linking the performance counter values to the power consumption. During runtime, counter readings can be plugged into these formulae to estimate the temperature. Such estimation involves just a handful of arithmetic operations and therefore imposes little overhead during system operation. The intermediate step of estimating the power can be skipped allowing the temperature to be directly estimated from the performance counters' values [22].

C. Classification Algorithms

Classification algorithms are an increasing focus of research interest in a variety of fields [23]. They use machine learning techniques to associate certain features of system-related data with certain system states. A traditional example from data analytics is to associate certain biomarker readings with disease, in expert medical diagnostic systems.

Using a classification algorithm to predict on-chip temperature has been proposed in [24]. Offline models of temperature are used and the real-time workload is also profiled extensively. The thermal predictor is invoked repeatedly with a given period. Two thermal states of a core are defined: overheated and normal. Simulation or modeling data are used to generate a correspondence between instances where the core becomes overheated in the next thermal predictor period given the current set of workload features. A Support Vector Machines (SVM) [23] approach is used to automatically "discover" the relationship between the workload feature set and the current state on the one hand and becoming overheated on the other. SVMs can then be employed to use such a offline-discovered relationship to make predictions during regular system operation. If a particular workload is predicted for the core over the next thermal management period, then task rescheduling/reassignment or other methods can be used to mitigate the problem.

D. Neural Networks

Neural networks can be trained offline, using the predicted workload of the system, and then used online to estimate (and predict) core temperatures. The effectiveness of this approach obviously depends on the choice of the input parameters on which training is done. Note that in every instance, the set of input parameters is just a synopsis of the thermal state of the system; the question is one of balancing the size of the input parameter set (and the resulting computational complexity of the predictor) and its ability to capture the thermal characteristics of the system.

An instance of training and then using neural networks to predict temperature can be found in [25]. The inputs taken by this network are the average and maximum power consumed by a core as well as the temperatures of the neighboring cores. There are one hidden, and one output, layer in the neural network studied in [25], with a total of seven neurons.

E. Comparison

Of these four approaches, using sensors and counters are the best evaluated to date. Techniques based on classification and on neural networks are much more recent and appear to require more validation before being used with confidence in a wide number of applications.

The number of thermal sensors, their positioning on the chip and the extent to which they are calibrated are all factors which determine the usefulness of the sensor approach to estimation. With performance counters, it is important to obtain a well-calibrated relationship between the counter values and the temperature at various positions on the chip.

V. Heat Flow Equation

Heat flow is typically modeled using a linear first-order differential equation. Power that is dissipated is expressed as heat, which contributes to increasing temperature and to heat flow. We approximate loci of power dissipation by *nodes*: these are the limits of resolution of the model. The amount of heat energy required to increase the temperature of a node by one degree Celsius is its *thermal capacitance*, C. The rate of cooling is determined by the difference between the temperature of a node and that of its neighbors as well as the *thermal resistance*, R, between them. The ambient temperature is usually modeled as a single node; generally, it is assumed to be at a constant temperature, unaffected by heat flows (in other words, the ambient is modeled as having infinite capacitance). We have already noted that implanted medical devices are an exception to this, where the ambient is the surrounding living tissue.

Denoting the temperature of node i at time t as $T_i(t)$, the thermal resistance between nodes i and j by $R_{i,j}$, the thermal

capacitance of node i by C_i, and the power consumption of node i by $\omega_i(t)$, the differential equation governing the heat flow is

$$\omega_i(t) = C_i \frac{dT_i(t)}{dt} + \sum_{j \neq i} \frac{T_i(t) - T_j(t)}{R_{i,j}} \tag{4}$$

The widely used Hotspot software for estimating processor temperatures is built around this equation [26].

Two comments are in order. First, the heat flow equation is linear. This linearity is of technical convenience, as we shall see. Second, there is the question of how to break down a given region of silicon into nodes to maintain model fidelity.

A. Linearity of the Heat Flow Equation

1) Accelerating the Heat Flow Solution: The linearity of the differential equation (4) can be exploited to speed up its solution by borrowing results from the field of linear control systems [27]. Note that we can write this set of equations in matrix form as follows:

$$\mathbf{C}\frac{\mathbf{dT}(t)}{dt} = \mathbf{D}\mathbf{T}(t) + \omega(t) \tag{5}$$

where $C = [c_{i,j}]$ such that

$$c_{i,j} = \begin{cases} C_i & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

$D = [d_{i,j}]$ such that

$$d_{i,j} = \begin{cases} -\sum_{k=1}^{N} R_{i,k}^{-1} & \text{if } i = j \\ R_{i,j}^{-1} & \text{if } i \neq j \end{cases}$$

and $\omega(t)$ is the vector of power inputs to the various nodes at time t. Using $\mathbf{T}(t)$ to denote the temperature vector, we can write this expression in matrix form:

$$\frac{d\mathbf{T}(t)}{dt} = \mathbf{C^{-1}}\mathbf{D}\mathbf{T}(t) + \mathbf{C^{-1}}\omega(t) \tag{6}$$

which readers familiar with basic control theory will recognize as the state equation of a linear system with state vector $\mathbf{T}(t)$ and input vector $\omega(t)$.

Now, the power input is usually only specified at sampled instants which are multiples of some sampling interval, Δt. The standard assumption is that the power consumption is constant over the interval between sampling instants; if Δt is small, this assumption is sufficiently accurate for all practical purposes. This corresponds to the model of a digital, linear first-order feedback system, with zero-order control [28]. From the theory of such systems, we can write the temperature vector at the $n-$th sampling instant (i.e., time $n\Delta t$) as:

$$\mathcal{T}(n) = \mathbf{A}\mathcal{T}(n-1) + \mathbf{B}\omega(n-1) \tag{7}$$

where

$$\mathbf{A} = e^{(\mathbf{C}^{-1}\mathbf{D}\Delta t)}; \quad \mathbf{B} = \int_0^{\Delta t} e^{(\mathbf{C}^{-1}\mathbf{D}(\Delta t - s))}\mathbf{C}^{-1} ds$$

By iteratively applying (7), we can write for any positive integer k:

$$\mathcal{T}(k) = \mathbf{A}^k\mathcal{T}(0) + \sum_{i=0}^{k-1} \mathbf{A}^{k-i-1}\mathbf{B}\omega(i) \tag{8}$$

If the power inflow is constant (and denoted by ω_c), we can write an expression for the steady-state temperature vector:

$$\mathbf{T}_{ss} = (\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}\omega_c \tag{9}$$

Exploiting the linear property of the heat flow equation significantly reduces the time complexity of the solution to the heat flow model: the TILTS software package owes its improved speed to this [29].

2) An Upper Bound Approximation: The linearity of heat flow can also be used to obtain an upper bound on the temperature of each node [30], [31]. Return to equation (6); the temperature vector can be written in matrix form as

$$\frac{d\mathbf{T}(t)}{dt} = \mathbf{F}\mathbf{T}(t) + \mathbf{G}\omega(t) \tag{10}$$

where $\mathbf{F} = \mathbf{C^{-1}}\mathbf{D}$ and $\mathbf{G} = \mathbf{C}^{-1}$. The solution of this system of equations is well-known:

$$\mathbf{T}(t) = e^{\mathbf{F}t)}\,\mathbf{T}(0) + \int_0^t \mathbf{H}(t-s)\omega(s)ds \tag{11}$$

where $\mathbf{H}(t) = e^{(\mathbf{F}t)}\mathbf{G}$.

The first term is the impact over time of the initial state. The second term is the sum of the impact of the power dissipation (since time 0) at each node. This second term is the sum of the contributions of each of the N nodes in the system. In classical control theory terms, $\mathbf{H}(t)$ is the *impulse response matrix*. That is, if τ seconds ago, there was an impulse of value δ_k at node k, its impact now (at time t) at node i is $H_{k,i}(t-\tau)\delta_k$. Integrating over all previous times yields the impact at time t at node i of the power consumed over the past at node k. Since the system is linear, the contribution of each of the neighbors is added to its own to find the aggregate contribution to the temperature at node k.

To find an upper bound, suppose $H_{k,i}^{\max}$ is the maximum value of $H_{k,i}(t)$ for all t. Let $\omega_i^{\max}$ denote the maximum power consumed by node i. Then, the maximum contribution of node k to the temperature of node i is upper-bounded by $H_{k,i}^{\max}\omega_k^{\max}$. Denote by $\omega^{\max}$ and $\mathbf{H}^{\max}$ the vector of maximum powers and the matrix of the maximum impulse terms, respectively. It then follows that the temperature vector is upper-bounded by

$$\mathbf{T}^{\max} = e^{(\mathbf{F}t)}\mathbf{T}(0) + \mathbf{H}^{\max}\omega^{\max} \tag{12}$$

The tightness of this upper bound will obviously depend on the power consumption curves. The upper bound has been calculated for each node i on the assumption that every other node's power consumption is timed precisely to heat up node i to the maximum extent. Such power consumption is obviously linked to the activity in each of the nodes, i.e., to the workload in each node; when this is done, we can come up with worst-case workload traces which can be simulated to yield the worst-case temperature at each node [30].

3) Additive Effects of Individual Tasks: The linearity of the heat-flow equation allows us to separately calculate the temperature impact of individual tasks and then add them up. This can often simplify analysis and reasoning considerably. Since the composition of two linear functions is itself a linear function, this benefit also carries over to any performance

measure which can be expressed as a linear function of the temperature.

As an example, consider the Accumulated Thermal Impact (ATI) measure mentioned in Section III. Let us consider a coarse-granularity model, where the entire processor core is modeled by a single node [7]. Denote by $ATI_{i,s_i}(t)$ the ATI of an individual iteration of periodic task i being run at speed s_i; its period is denoted by P_i. Since an individual iteration is finite and its temperature impact decays exponentially with time, this quantity has a limit, $ATI_{i,s_i}(\infty)$. If there are n tasks in the system (numbered 0 to $n-1$), then linearity can be exploited to show that $\sum_{i=0}^{n-1} ATI_{i,s_i}(\infty)/P_i$ is a lower bound on the maximum temperature of the processor[2]. Note that since linearity allows us to separately add up the contributions of the individual tasks for the ATI limit, $ATI_{tot}(\infty) = \sum_{i=0}^{n-1} ATI_{i,s_i}(\infty)$ is not dependent on the actual task schedule (i.e., on *when* each task is run). Furthermore, since the ATI of a task is obviously directly related to its total energy consumption, we can show (again invoking the linearity of the heat flow equation) that any selection of task execution speeds s_i which minimizes the total energy consumption of the task set also minimizes $ATI_{tot}(\infty)$. Thus, the entire set of dynamic voltage and frequency scaling techniques (described later in Section VI-C). used to minimize energy consumption also automatically minimizes $ATI_{tot}(\infty)$.

B. Model Granularity

In the above discussion, we have treated the nodes as abstract entities, consuming a certain amount of power (which may be zero in the case of passive elements like heat sinks), having a given thermal capacitance and whose thermal linkage to other nodes is modeled by thermal resistance. The question of how large a silicon area should constitute a node is not clear. To a large extent, it depends on the input information available about the workload. If this information is limited to the total power consumption at each core, then we have no means to divide it down among its various functional units, and have to treat the entire core as a single node.

A more nuanced picture emerges when we are given the workload. The workload can be run through an architecture simulator (such as GEM5 [32]) and the activity of each of its components (functional units, buffers) recorded. The power consumption caused by such activity can be estimated by a software tool (such as McPat [33].) This allows for a finer-grained representation of activity in the core and allows us to model temperature variations inside the core, information that would be lost if we treated the entire core as a single node. This approach is taken in, for example, [22], [34]. A wide variation in temperatures between subunits is observed when fine-grained temperature models are used.

Most of the research published so far in thermal-aware scheduling uses a coarse-granularity model, with a core being modeled by a single node. The system of differential equations governing heat flow reduces, then, to a single equation and analysis is greatly simplified. Rather than dealing with the vector representing power consumption at each of the various units in the processor core, we only work with a scalar function of the power consumption at the entire processor. Further simplification is possible, and conclusions are often drawn, assuming steady power consumption over time.

Two questions must be addressed when considering the impact of model granularity:

- How valid is the single node assumption, i.e., are temperatures, in practice, really more or less uniform across a chip?
- If temperatures are not, in fact, spatially uniform, then how does this deviation from uniformity affect the modeled impact of thermal scheduling techniques using the various optimization criteria described in Section III?

In practice, as chip temperature maps show, on-chip temperatures vary considerably from one location to another. Typically, caches are among the coldest units of all; while they may consume a considerable amount of energy in the aggregate, this energy is typically dissipated over so wide an area that the cache energy density is very low. At the other end of the range are the integer and floating-point registers and functional units; these are typically among the hottest elements. Of course, all this depends on the workload; for instance, workloads heavily dominated by integer operations will leave floating-point registers cold.

We now turn to the second question, i.e., how using the coarse-grained model affects our evaluation of the thermal management techniques.

If the goal is to keep the maximum temperature below a given limit, then modeling the entire chip as a single node will result in inaccuracies, since the peak average temperature is likely (as mentioned above) to be very different from the peak spot temperatures. Performance numbers and conclusions drawn from techniques using the peak temperature as the optimization criterion must therefore be treated with caution when evaluated with a coarse-granularity model.

If the criterion is ATI, then coarse-granularity models will provide quite accurate results. The reason is that the heat capacity (modeled as thermal capacitance) of the entire chip is the sum of the heat capacities of the individual subunits. The total energy consumed is the sum of the energy consumed at the individual subunits. From this and the linearity of the heat flow equation, we can conclude that the ATI value will be accurately estimated by modeling in the aggregate (which is what the coarse-grained model does). Furthermore, due to linearity, the contribution of each task to ATI can be added up after being assessed separately as a function of its dynamic power consumption (recall that static power consumption is a function of temperature and is folded into the heat-flow differential equation), its period, and its execution time. Thus, a task which has period τ_P, average dynamic power consumption ω^{dyn}, and execution time (per iteration) of e, will contribute $K\omega^{dyn}e/\tau_P$, where K is a constant of proportionality. Note that we can easily model multicore systems using the same approach: the only difference will be that the impact of power dissipation at core i on the temperature at core j has to be

[2]We assume here that each iteration of the same task i is run at clock speed s_i. The adjustment required when this is not true and different iterations of the same task are run at differing speeds is rather straightforward and left to the reader.

modeled for each (i, j) pair. This is captured by deriving pairwise constants of proportionality, $K_{i,j}$ [35].

If the criterion is TAAF, then coarse-grained models will likely not give accurate estimates. The reason is that TAAF is a nonlinear function of temperature. As noted in Section II-B, many failure modes follow the Arrhenius model and are exponentially accelerated with temperature. The average chip-wide temperature underestimates the temperature of some regions while overestimating that of others. The improvements in TAAF from having a region cooler by ΔT^o C are outweighed by degradations from having an equivalent area hotter by ΔT^o C; the greater the value of ΔT, the more the deviation from the average TAAF. Note that the extent to which this happens depends on the activation energy, E_a.

If the criterion is RTR, then coarse-grained models should be sufficient, so long as the heat spreader is efficient enough that the unit presents a uniform temperature to the surrounding environment. Recall that the principal concern underlying the use of RTR is the impact of processor heating on its surroundings.

The granularity of the thermal model also affects the estimated thermal time constant: this is analogous to the RC time constant in electrical circuits. The thermal time constant is an indication of how rapidly the temperature of the node rises in response to power dissipation. If a coarse-grained model is used, the high value of the thermal capacitance usually leads to the thermal time constant being in the order of milliseconds, if not higher. However, thermally induced damage occurs as a result of *localized* heating for which one can expect the time constant to be much lower. The point behind this remark is that designers should not take too much comfort from large nodal thermal time constants; these are likely to be an artefact of the model granularity rather than representing low-level physical impact on the chip.

VI. Thermal Control Options

A. The Heat Flow Equation Revisited

In the preceding discussion, we assumed that the power consumption is known; the implicit assumption is that it is largely independent of temperature. However, power consists of a dynamic and a leakage component. While the leakage power is typically modeled as an exponentially increasing function of temperature, it has been shown that for the range of typical temperatures, leakage power can be modeled quite accurately as increasing linearly with temperature [36], i.e., the leakage power at node i whose temperature at time t is $T_i(t)$, is given by $\alpha T_i(t) + \rho$ for some constants α and ρ. Such a linear approximation allows us to quite easily fold the leakage power into the heat-flow differential equation by writing $u_i(t) = \omega_i^{dyn}(t) + \alpha T_i(t) + \rho$ and making the obvious adjustments, where $\omega_i^{dyn}(t)$ is the dynamic power consumption caused by circuit switching [30]. This latter term now becomes the independent term in the differential equation; it can be controlled by scheduling tasks and processing resources. Over the rest of this section, we will assume that an entire processor core is modeled as an isolated, single node in our thermal model. The power and temperature vectors thus degenerate to scalars.

To see that the independent term can be controlled by scheduling, consider what makes up the dynamic power consumption. It is the power dissipated as a result of circuit switching and is proportional to the product of (a) the amount of activity in a cycle, expressed by the number of switches taking place in the same cycle, (b) the frequency of switching, and (c) the energy consumed per switch. This last quantity is proportional to the square of the supply voltage. Now, the maximum switching frequency can be roughly modeled as linearly rising with supply voltage (within acceptable voltage bounds). (This assumes that we are running the clock at its voltage-determined limit; it is easy enough to suitably modify the equation if we are not.) Putting all this together, the dynamic power is expressed as a cubic function of the switching frequency and linear in the number of switches in the same cycle. Denote the activity factor by $a(t)$; that is, $\omega_i^{dyn}(t) = Ka(t)f^3(t)$ where $f(t)$ is the clock frequency at time t [37]. If we know the constant of proportionality, K, we can plug this back into the heat flow differential equation to obtain the dependence of the temperature on the activity of the circuit, its switching frequency, and the circuit leakage parameters. The total power consumption is therefore given by $P(t) = \alpha T(t) + \rho + Ka(t)f^3(t)$.

We can now write the temperature differential equation as follows:

$$C\frac{dT(t)}{dt} = -\frac{T(t) - T_{amb}}{R} + P(t) \tag{13}$$

where T_{amb} is the ambient temperature, and R, C are the thermal resistance and capacitance, respectively, associated with the processor.

What is the relationship between the activity factor $a(t)$ and the amount of useful work done? In the simplest processors, it is roughly linear. However, modern processors employ a large number of tricks to speed up execution; most of these involve speculative execution [38]. Examples include branch prediction and out-of-order execution of instructions. In such a case, a certain portion of the activity is being carried out in the hope that it will be useful based on a guess by the system as to the correct execution path. Modern processors owe much of their speed to being able to guess at a high success rate. However, the more aggressive the speculation and the deeper the processor pipeline, the lower the guess success rate tends to be. For simplicity, in our discussions below, we will assume that the useful work is linearly related to the activity level; the reader should, however, always keep in mind that this is an approximation. The actual relationship is a function of the processor architecture, the quality of the compiler, and the application software being run.

B. Power Gating

Power gating is one of the simplest thermal control schemes. It requires only that we can turn the processor on and off. That is, we run the system at a certain frequency for a certain amount of time, then turn it off to cool down before turning it on again. Consider a setup where we turn on the processor

(at some given frequency) when it reaches a temperature of T_0, run it until it heats up to temperature T_1, switch it off (meaning its power consumption goes to zero) until it cools back to T_0, and continue this cycle indefinitely. Suppose we have a roughly constant activity factor, i.e., $a(t) \equiv a_0$ and a constant frequency, f. We can now solve the differential equation (13) to find the duration for which the system can be kept on, denoted by τ_{on}. Then, power is turned off and the processor is allowed to cool back to T_0: the period for which it is off is denoted by τ_{off}.

Solving (13), we obtain the on and off times. For notational convenience, define $T_\infty = \frac{T_{amb}+(\rho+Kaf^3)R}{1-\alpha R}$ and $\xi = \frac{1-\alpha R}{RC}$. T_∞ is the steady-state temperature if we never switched off the processor. We can then write:

$$\tau_{on} = \begin{cases} -(1/\xi)\ln\left(\frac{T_\infty - T_1}{T_\infty - T_0}\right) & \text{if } T_1 \le T_\infty \\ \infty & \text{otherwise} \end{cases} \quad (14)$$

$$\tau_{off} = \begin{cases} -RC\ln\left(\frac{T_0 - T_{amb}}{T_1 - T_{amb}}\right) & \text{if } T_{amb} < T_0 < T_1 \\ 0 & \text{otherwise} \end{cases} \quad (15)$$

Gating is not free: there is a certain time consumed by it while the processor is active but not doing any useful work. Let the time for each switch be τ_{gate} (assume that the turn-on and turn-off times are roughly the same). Then, for each cycle of duration $\tau_{on} + \tau_{off}$, the system is able to spend $\tau_{on} - 2\tau_{gate}$ on doing useful work.

Figure 1 shows the impact of selecting T_0 and T_1. As T_0 gets closer to T_{amb}, the off-interval τ_{off} increases exponentially; similarly as T_1 approaches T_∞, the on-interval τ_{on} increases exponentially. Note, however, that increasing either T_0 or T_1 results in greater thermal damage. Both trends together contribute to the duty cycle, which is the ratio of the on time to the sum of on and off times.

One variation of this approach notes that the leakage is a function not only of temperature but also of supply voltage [39]. Since supply voltage is linked to the maximum clock frequency, we can quite easily fold this parameter also into our main heat flow equation.

C. Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency Scaling (DVFS) uses the relationships mentioned above. If we have the technical capability to switch supply voltages and clock frequencies, then we can tune the performance of the system to minimize the energy consumed while still meeting task deadlines.

The question is whether, in a multicore system, we have the ability to carry out DVFS independently in each core or whether the scaling can only be done chip-wide (meaning that each core on the chip runs at the same voltage and clock frequency). Obviously, supporting the former capability adds complexity to the chip; such capability is not always available. Having to run all cores at the same voltage/frequency combination reduces our flexibility considerably, since we have to run the overall clock at a frequency that ensures that *every* core meets its workload hard deadlines.

If we use DVFS, the frequency is a parameter that can be controlled to change the thermal behavior of the system and the speed of processing. The activity factor, $a(t)$, depends on how the processor architecture interacts with the workload [37], [39]. For example, if we use a very simple linear pipeline issuing at most one instruction per cycle and not capable of any Instruction Level Parallelism (ILP), then $a(t)$ is controlled by the workload. Controlling $a(t)$ is then a matter of deciding, by scheduling, the tasks running at any given time. On the other hand, if we use high-end modern processor cores, we have multiple instructions (potentially) being fetched, executed, and retired, per clock cycle. Controlling the level of ILP by, for example, limiting the number of instructions fetched per cycle, gives us another means to control the activity factor, $a(t)$.

A complementary approach to DVFS has been recently studied [34]. The idea behind this is that the activity factor varies with time, even for the same task. Many applications have distinct phases of execution with widely varying levels of ILP [37], [40]. If we have a total amount of schedule slack that we can "spend" by slowing down the processor, we get better savings in energy (and in temperature) by preferentially slowing down the high-activity regions of code. A very simple calculation suffices to show what improvements might be possible, at least to a first order. For example, suppose slowing down reduces power consumption by a factor α_p and increases execution time by a factor of α_e. (That is, the ratio of the decreased to original power consumption is α_p and the ratio of increased to original execution time is α_e.) Let the runtime of the task at high voltage be τ, of which τ_h is in a high-activity region and $\tau_\ell = \tau - \tau_h$ is the rest. Let ω_h, ω_ℓ be the average power consumed at high voltage in the high-activity and low-activity regions, respectively; the average power consumption is $\omega_{av} = (\tau_\ell\omega_\ell + \tau_h\omega_h)/\tau$. Each second that we devote to slowing down the high-activity region saves us energy $\omega_h(1 - \alpha_p\alpha_e)$; while applying it at random saves us just $\omega_{av}(1 - \alpha_p\alpha_e)$, on the average. (Note that since power consumption drops much faster than the execution time increases with voltage scaling, $\alpha_p\alpha_e < 1$.) If ω_h/ω_{av} is large, there are considerable savings to be had from focusing our slowdown efforts on the high-activity region (assuming that τ_h is large enough to make a difference).

Implementing such a policy requires advance profiling of each task, to determine its high-activity regions. Such parameters as the rate at which instructions are retired can be used to detect when a program is in its high-activity region. The effectiveness of this approach will depend on how repeatable the task characteristics are: clearly, if the high-activity fraction varies significantly from one iteration of the task to the next (due, for example, to changes in input parameters), this technique will not work well.

D. Architecture Adaptation

Modern pipelines are complex and use multiple techniques to increase the number of instructions that can be executed in parallel. Such techniques increase the amount of activity on the chip and consequently, the power consumption.

Adapting the architecture, by limiting the fetch rate or by switching off components of the core, allows us to control the power consumption at the cost of increased execution time.

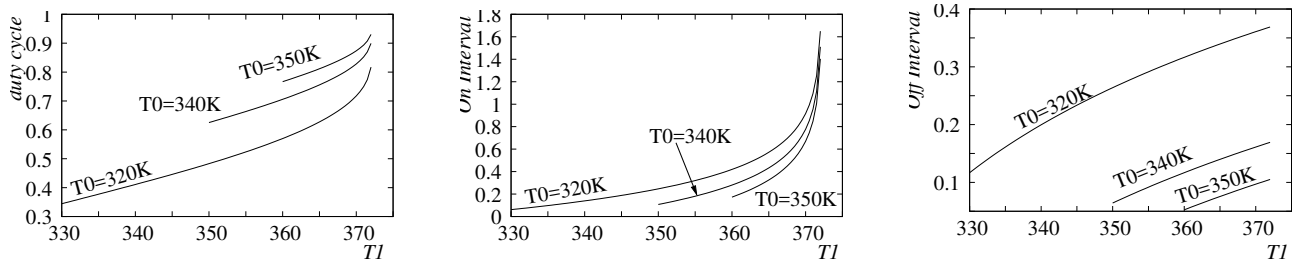

Fig. 1. Power Gating: Impact of T_0 and T_1 Selection ($T_{amb} = 300K$, $\rho = 0.1; K = 10E-6; a = 1; f$ =1 GHz; $R = 0.36; C = 0.8; \alpha = 0.001$)

This has been studied, for example, in [41], [42]. Various approaches can be taken to adapt the activity:

- *Fetch Unit:* The number of instructions fetched per cycle can be varied. Furthermore, we might alternate idle cycles between consecutive fetches, for example, we might idle the fetch unit for one or two cycles after every active fetch cycle.
- *Buffers:* Processors typically contain multiple buffers. By reducing the number of available (to the processor) entries in such buffers, we can reduce the level of activity. For example, the reorder buffer size can be varied: the size of the reorder buffer limits the number of instructions in flight at any given time. We can also change the size of the load-store queue; this reduces the number of load/store operations that can be supported and throttles back the processing rate.
- *Functional Units:* There are typically multiple integer and floating-point functional units available. By turning off some of them, we can reduce the rate at which arithmetic and logical operations are carried out.

The more complex the pipeline, the more options are available for architecture adaptation. So long as control capability exists to carry out such adaptation, we can do so with minimal overhead and no complicated additional software requirements.

Architecture adaptation can be carried out in a focused manner. That is, individual programs can be profiled and a cost/benefit analysis can be carried out as to the appropriate level of speculative execution that returns good results (and beyond which processor activity is largely wasted). Such a profiling action can be used to adapt the architecture keeping the task in mind; for different tasks, different adaptations may yield the best cost/benefit ratio. For some case studies, see [41], [42].

E. Greedy Slack Deployment

If we have a purely periodic task set with some slack available in the schedule, one approach is to deploy the slack so that some target temperature is maintained. One way to do this is presented in [37]. Recall from Section V-A3 that a lower bound of the temperature in a periodic task set running on a uniprocessor is given by $\sum_{i=0}^{n-1} \frac{ATI_{i,s_i}(\infty)}{P_i}$. We can set that up as a target temperature. So long as there is slack in the schedule (with respect to each task), we have the freedom to schedule *any* task without risk of missing a deadline. Once the slack of some task has been reduced to 0, we are constrained in which task we can execute. Different tasks have different activity levels, i.e., they differ in the amount of heating they cause. If we have information about the estimated activity level of each task (this can be obtained – to some level of accuracy – by profiling the task on the family of anticipated input sets), we can use such information to keep the temperature close to the target without missing deadlines. One heuristic for this is to do the following [37]:

- If each task has slack available (at least equal to the duration between successive invocations of the scheduler), then greedily pick whichever task (or none) will keep the temperature of the core closest to the target until the scheduler is invoked again (likely as a periodic task in its own right).
- If one or more tasks has no slack available, then use the traditional Earliest Deadine First (EDF) algorithm to schedule processor activity.

Typically, there is a considerable amount of slack in the schedule of cyber-physical systems. There are two reasons for this. First, we do not often schedule tasks to fully utilize the available capacity. (For example, some reserve is often kept to provide time redundancy against transient failures or to permit the processor to assume some of the workload previously assigned to a failed processor [43].) Second, tasks usually run to much less than their worst-case execution times, and an early-finishing task releases slack into the schedule (with respect to tasks with lower priority than itself).

F. Task Assignment Using Complementary Tasks

In a multicore system, tasks can be assigned based (in part) on their thermal characteristics. At a coarse granularity, tasks may be classified as "hot" or "cold," and each core might be assigned a roughly balanced mix of hot and cold tasks. When scheduling, we can try to interleave hot and cold tasks, under the constraint that all hard deadlines must continue to be met.

A finer-granularity, and evocatively named, "Heat-and-Run," approach has also been suggested (albeit in a non-CPS context) [44]. This uses the fact that heat flow on a chip is typically much more efficient in the vertical direction (i.e., through the heat sink) than in the horizontal (to a neighboring part of the silicon). If we group tasks so that they exercise different parts of the chip (e.g., one task in the group intensively uses the floating-point registers and functional units while another restricts itself mostly to using the integer side), then complementary areas of the chip are heated up. Then, we can assign such task groups to an individual core, allow the

core to heat up in response to the load, and then "run," i.e., migrate the entire task group to another core while the first one cools down.

G. Task Assignment Using Quotas: 3D Chips

As technology advances, 3D chips are likely to become more prevalent. These consist of cores stacked one above the other in layers. (2D VLSI is already multilayered; here, we refer to layers of cores, each of which is itself constructed out of multiple metal and other layers.) This changes the heat flow pattern between cores. In a 2D arrangement, while there is obviously heat flow from one core to another, the dominant heat flow is from a core to the heat sink. In a 3D arrangement, where cores are stacked one atop another, the dominant heat flow is still through the heat sink; however, heat from a core far away from the heat sink will have to flow through other cores to get there. There is, therefore, much greater thermal coupling between cores in a 3D arrangement than there is in 2D.

Consider, for example, an arrangement where we have a 4-layer system. If the heat sink is at the bottom, then the heat generated by the top layer cores is mostly dissipated through the 3 cores below them to the heat sink (while there is some heat flow in other directions as well, we are focusing here on the dominant component). In such a case the "ambient" temperature as seen by a core is the temperature of the core directly below it.

Any task assignment on a 3D system must therefore take into account the vertical position of each core. One way of doing this is the idea of *thermal size* [45]. In what follows we will follow [45] in assuming a single heat sink at the bottom which governs the dominant heat flow. The thermal size of a node is the difference between its steady-state temperature and that of the core below it (or the heat sink if this is a bottom-layer core). The thermal-size approach consists of (a) assigning a thermal size quota to each layer, and (b) assigning workload to each core based on its thermal size.

Denote by $r_{j,j-1}$ the thermal resistance between a layer j core and the core below it (or the heat sink if this is the bottom layer). (We count layers from the bottom up: the bottom layer is layer 1 and the heat sink is layer 0.) The resistance on the dominant heat flow path from a core in layer i is therefore given by $R_i = \sum_{j=1}^{i} r_{j,j-1}$. Assume we are given a maximum thermal size for the entire stack, ψ_{stack}; this is the sum of the thermal sizes of each layer in the stack: $\psi_{stack} = \sum_{j=1}^{\ell} \psi_j$ where there are ℓ layers in all. Several ways of assigning thermal size now suggest themselves:

- Average Assignment: Assign each layer an equal thermal size:
$$\psi_i = \psi_{stack}/\ell,\ \forall i = 1, \cdots, \ell$$
- Balanced Assignment: Make an assignment by considering the resistance of the path from each core to the heat sink:
$$\psi_i = \psi_{stack} \frac{R_i}{R_1 + \cdots + R_\ell},\ \forall i = 1, \cdots, \ell$$
- Proportional Assignment: Assign by considering the thermal conductances of the path from each core to the heat sink:
$$\psi_i = \psi_{stack} \frac{R_i^{-1}}{R_1^{-1} + \cdots + R_\ell^{-1}},\ \forall i = 1, \cdots, \ell$$

A task assignment can then be done by considering the peak power, π_i of each task i and considering its consumption of thermal size at layer j to be $\pi_i R_j$. (Note that if the core-to-heat-sink heat flow path is dominant, this quantity is the peak steady-state temperature caused by consuming π_i watts of power at a layer-j core.) We must ensure that the total thermal size consumption of all the tasks assigned to each core does not exceed its assigned quota, based on the assignment approaches mentioned above.

H. Task Migration

Task migration is an obvious way to control the loading on a core. It can be used either by itself or in conjunction with other approaches.

We have already encountered task migration in Heat-and-Run. There are two types of task migration: migration of a task before it starts execution, and migration of a partially completed task.

Migration of a task before it starts executing simply consists of moving the task executable to the target core and inserting it into the appropriate task queue. Before a task is migrated, we have to ensure that it does not overload the target core and lead to hard deadlines being missed. Standard techniques from real-time scheduling theory can be used for this purpose [46].

Migrating partially executed tasks requires the state of the task to be migrated. Depending on the size of that state and the on-chip bandwidth, this may or may not be expensive. Also, if virtual memory is used, this involves the movement of page tables.

The cost of migration depends on the structure of the system. For example, if we have multiple cores sharing a lower-level cache, then moving a task may not take much time. On the other hand, if we have a loosely coupled system with independent caches, moving the task can lead to significant delays and may not be worth the effort.

Deciding when to trigger a move is usually made based on a simple heuristic. (Note that in a real-time system, we always need to verify that the migration does not cause any critical task to miss its deadline.) For example, in [47], a temperature threshold, T_{lim}, for each core is prespecified along with a parameter ξ. The task migration code is invoked periodically; if the temperature is judged to fall, over the subsequent period, in the range $[T_{lim} - \xi, T_{lim}]$, then migration is triggered. In [25], there are two procedures suggested. In the first, the steady-state temperature of each core is computed based on its power consumption. When neighboring cores differ in such temperature by more than a threshold, workload can be swapped between them, with the hotter core exchanging a hotter (i.e., more power-consuming) task for a cooler one. The second procedure in [25] uses current or predicted temperatures instead of steady-state ones; this allows one to capture the

transient effects of a varying workload. A two-phase process is carried out to find the best hotter/cooler task pair to swap between a hotter and a cooler core.

It is possible to adaptively adjust the temperature threshold at which migration is triggered [48]. The system can increase the threshold if migrations are too frequent; it can relax the threshold back down if this is no longer the case.

In [49], cores that are warmer than their immediate neighbors consider exchanging tasks with them. The aim is to carry out an exchange that results in the new arrangement delivering a lower temperature. Note that the same task can be involved in multiple migrations, i.e., a task which was exchanged from core 1 to core 2 can then be moved on in an exchange from core 2 to core 3, and so on.

In general, picking a target core to migrate to is also usually done using a heuristic. The obvious approach is to migrate to the coldest core which has the spare capacity to assume this additional workload. However, this may not always be the best target, since it may be physically close to warm cores which may be heated up as a result. We might, instead, pick a target core which (given the activity in its surroundings) will take the longest time to get heated up [48].

I. Evaluation

The thermal management technique selected depends on the circumstances. Power gating is easy to implement: it requires only a temperature monitor. However, the on and off temperatures have to be selected carefully, keeping the needs of the application in mind: we have to effectively balance the available processor utilization against the thermal damage that is allowed. Also, simple power gating can result in rapid temperature swings, which can elevate the failure rate. One great advantage of power gating is that it is orthogonal to the other approaches. When feasible, power gating can be used as a backup safety mechanism, simply to prevent excessive heating, relying on other (more sophisticated) techniques to reduce thermal damage.

Voltage scaling requires us to have multiple voltage levels available. As technology advances, the maximum supply voltage decreases. The reduces the range of voltages available; as this happens, the scope for voltage scaling will naturally decrease. Further, if task ILP is to be used in managing voltage scaling, these tasks need to be profiled, either in advance or during execution.

Architecture adaptation can be used in combination with the other approaches. However, it requires low-level control of the architecture. Also, it is only applicable when the pipeline is complex enough that there are multiple configurations offering a sufficiently wide range of performance/thermal options.

Greedy slack deployment and complementary tasks are based on the same idea: separate high-heat intervals by low-heat intervals. Both require advance profiling of tasks. Further, complementary tasks may not always be available from among the set of ready tasks.

Task migration is theoretically appealing but requires a system architecture which facilitates migration. In many instances, migration imposes considerable overheads in aligning the process state appropriately. In a 3D architecture, migration may be combined with quotas. Quotas require extensive advance profiling when used alone; however (at least theoretically) a learning approach which learns task characteristics amy be used instead. An initial assignment can be adjusted with task migration as better information is obtained about task characteristics.

VII. Proactive and Reactive Management Approaches

The various thermal control options can be applied either in a *reactive* or *proactive* way. Reactive means that we monitor the temperature and react when it gets too high. Proactive means that we try to prevent the core temperature from rising too much in the first place. A purely proactive scheme, which has no reactive component at all (and operating as an open-loop system), will need to use highly conservative assumptions as to the impact of execution on temperature. For this reason, to be efficient, it is likely that proactive schemes will need to have some component that is reactive, i.e., involve feedback. To that extent, the term *proactive* as used to describe thermal scheduling algorithms is somewhat misleading.

A. Proactive Management: Speed Selection

Perhaps the simplest proactive approach of all is to calculate if a given task schedule will breach thermal limits on any core, and if so, try to adjust the schedule to meet thermal constraints.

The key question is at what frequency-and-voltage combination we can run a core (without stopping) without overheating it. We shall concentrate here on the single-core case. Returning to Equation (13) and setting the differential $\frac{dT(t)}{dt} = 0$, we get the steady-state temperature: $T_{ss} = T_{amb} + P_{ss}R$. Here, we assume a constant power consumption P_{ss}, at steady-state frequency f_{ss} and activity factor a_{ss}. Since $P_{ss} = \alpha T_{ss} + \rho + K a_{ss} f_{ss}^3$, we have the result

$$f_{ss} = \left(\frac{T_{ss}(1-\alpha) - T_{amb} - \rho}{K a_{ss}} \right)^{1/3} \tag{16}$$

If the activity factor is variable, then we can use its upper bound in place of a_{ss} in the above expression to get a safe frequency at which to run the processor.

B. Reactive Management: Using Feedback Control

Feedback control theory can be used to determine the workload that can be executed under the constraint that the processor temperature should be mostly at, or below, a target value.

Much of the theory of feedback control is devoted to calculating input values to ensure that a given controlled plant follows a specified state trajectory with little error [50], [51].

For our purposes, the state is the temperature, and one aim might be to keep the processor temperature as steady as possible, at a relatively safe level. The control input is the power consumption. A discussion of the control techniques used to calculate the optimal input to minimize the tracking error of a feedback control system is outside the scope of this

paper; over the past several decades, researchers have built up a formidable array of techniques to do this: these are described in any book covering optimal control theory, e.g., [50], [51].

Unlike in traditional control theory, we do not really have freedom in selecting the control inputs. The power consumption depends on the task being executed. The principal means we have of controlling power consumption are (*a*) Dynamic Voltage and Frequency Scaling (DVFS) and (*b*) duty cycling. Earlier, we saw how DVFS would work. If DVFS is not available, duty cycling can be used. In this approach, the processor is active for only a fraction of the time. For example, we can keep the processor active only for τ_a seconds during each period of τ_p seconds, for a duty cycle of τ_a/τ_p. We can control the *average* power consumption by adjusting the duty cycle. Given that the thermal time constant of a chip is in the order of milliseconds, that may be all that is required for thermal purposes so long as τ_p is kept small. (Keep in mind, however, that there is an overhead associated with duty cycling; the smaller the value of τ_p, the greater the fraction of time spent turning activity on and off.)

Given a set of prescribed performance levels (recall the modes described in Section III), we can then link the performance level possible for each average power consumption. A design framework built on this principle is described in [11].

Another control-theoretic approach involves using processor utilization as the control [52]. Two controllers are used: a thermal controller and a utilization controller. The thermal controller assesses the current temperature and adjusts the target core utilization to prevent overheating. Since the thermal time constant of the chip (the implicit assumption here is that the entire core is modeled as a single node in a coarse-grained thermal model) is large, the claim is that this controller only needs to update its utilization targets at a relatively low frequency. The utilization controller has the job of tracking the target core utilization set by the thermal controller. It is responsible for dispatching tasks for execution. Note that one can incorporate DVFS into the inner loop quite easily by redefining utilization to be the number of useful computational cycles delivered in a unit interval. Traditional control-theoretic approaches can be used to select the appropriate rate at which each of the two controllers must be run. Note also that one can extend this approach to a more fine-grained thermal model. For example, if we have temperature readings from individual parts of a core (e.g., integer and floating-point register files, functional units), we can use the maximum of these temperatures as the parameter to be kept below some limit and set the utilization appropriately. This is a core-by-core control mechanism; when the entire computational workload is running on multiple cores, we will need to assign and reassign tasks based on the target utilization of each core. Furthermore, in a multicore setup, we will need to account for thermal interactions between neighboring cores.

VIII. Evaluating and Ensuring Schedulability

Schedulability checks are a vital part of cyber-physical systems. We need to be sure that the computational platform is capable of executing the critical workload required to provide commands to the controlled plant actuators in a timely and correct fashion. In this section, we consider how thermal considerations impact such schedulability checks. Before doing so, we provide a brief description of how schedulability checks are done in traditional real-time systems (without thermal considerations).

We will focus on uniprocessor scheduling of a set of periodic and independent tasks, each of whose relative deadline equals the task period. That is, a given iteration of a task has to be completed before the next iteration of the same task arrives. A few remarks are provided later on about how to deal with multiple-core platforms and interacting tasks.

For simplicity, assume that we are running at a fixed frequency, f. Denote by $\mu(\tau)$ the minimum amount of service (measured in computational cycles or execution time at a standard frequency) provided by the computational platform over an interval of time, τ.

Now, consider the demand induced by the workload assuming a set of independent and periodic tasks. There are two standard uniprocessor scheduling algorithms used in real-time systems: the *Rate Monotonic* (RM) and the *Earliest Deadline First* (EDF) algorithms [53], [8]. In RM, a periodic task's priority is inversely related to its period, i.e., the smaller its period, the greater its priority. In EDF, a task's priority is related to the closeness of its absolute deadline: the closer a task's deadline is to expiring, the greater its priority. Hence, in RM, we will always pick for execution the lowest-period task in the ready queue. In EDF, we will pick the earliest-deadline task in the ready queue. Tasks are preempted whenever a higher-priority task arrives; we follow scheduling-theory tradition here by assuming that such preemption costs are negligible. Now, there are exceptions and work-arounds for regions where tasks cannot be preempted: these consist of calculating the maximum time for which a lower-priority task can block a higher-priority task (for an example of recent work in this area, see [54]). Other recent work in scheduling for real-time includes limited preemptive scheduling [55], mixed criticality systems [56], federated scheduling approaches [57], varying period approaches [58] and parallel scheduling [59]. In all these cases, the scheduling algorithm picks which task is to be executed on each processor or core. The number of computational cycles that can be delivered by each processor depends on the thermal control mechanisms employed. Schedulability checks consist of evaluating whether the number of cycles delivered is sufficient to meet task deadlines given the priority structure imposed by the scheduling algorithm.

For concreteness, let us focus on the most commonly used scheduling algorithm in real-time systems: RM. Denote by $\nu_i(\tau)$ the maximum service required over any interval of duration τ by tasks of priority i and higher. We can use μ and ν_i to compute the maximum response time of task i. An iteration of task i can only execute when a higher-priority task is not ready to execute. Based on this, we can write down an expression for the maximum response time (time between release and execution completion) of such a task: such a time, τ_i^{resp}, is given by $\tau_i^{resp} = \min\{\tau : \mu(\tau) \geq \nu_i(\tau)\}$. Task i is schedulable so long as it responds by its relative deadline, which is its period, P_i. So, we need to check that $\tau_i^{resp} \leq P_i$

for all critical tasks.

It only remains for us to compute $\nu_i(\cdot)$. This function is maximized when an iteration of task i arrives at the same time as the iteration of every other higher-priority task. The maximum number of such iterations of any task j over any interval τ is given by $n_j(\tau) = \lceil(\tau/P_j)\rceil$. Hence, we must have $\nu_i(\tau) = \sum_{j=1}^{i-1} n_j(\tau)e_j + e_i$, where e_i is the worst-case execution time of task i. (Note that we traditionally number tasks in decreasing order of priority: task 1 is the highest priority task.)

How is this calculation affected when thermal considerations are introduced? Suppose we assume that the computational demand (represented here by $\nu_i(\cdot)$) is unaffected as it is determined by the needs of the cyber-physical plant being controlled. What is affected is $\mu(\tau)$, the minimum amount of computation that can be delivered by the system over any interval of duration τ. In a system whose activity is modulated by thermal considerations, we have to replace $\mu(\tau)$ by $\mu(\tau, T_0)$ which is the minimum amount of computation the system is guaranteed to deliver in any interval of duration τ which starts with the system being at initial temperature T_0.

To take a concrete example, consider the single reactive scheduling approach of [60], discussed earlier. Here, we run the system at any frequency up to when it reaches its temperature limit and then run it at f_E to ensure it does not exceed that limit. Hence, we must do the following:

- Solve the thermal differential equation (13) with initial condition corresponding to a specified initial temperature, T_0 to determine how long it can run at $f_{\max}$ before it reaches the temperature limit. Let this time be t_a.
- We now have two cases:
  - If $t \leq t_a$, then we are running at $f_{\max}$ throughout and $tf_{\max}$ cycles are delivered.
  - If $t > t_a$, then we run the first t_a time at $f_{\max}$ and the rest of the time at f_E; hence $t_a f_{\max} + (t - t_a)f_E$ cycles are delivered.

To evaluate schedulability under this scheme, we would therefore need to obtain an upper bound of the temperature at a given point in time, t, to be inserted into $\mu(t, T_0)$. Given the workload, this can be done: see [60] for details.

Another option is to work on the demand side. There are several ways in which this can be done. One is to reduce the order of complexity of the system being controlled. For example, if the controlled plant is modeled as an n-order linear system, we can use techniques such as Principal Component Analysis (PCA) to approximate it as an m-order linear system, where $m < n$. Calculating control inputs for this reduced order system can reduce the workload. The cost is likely to be a reduction in the quality of control.

Another approach is to adjust the amount of fault-tolerance applied. Life-critical cyber-physical systems must be extraordinarily reliable. For this reason, fault-tolerant techniques are used to keep their catastrophic failure rate within very low levels (e.g., 10^{-9} for a 10-hour flight in a fly-by-wire airliner). Fault-tolerance consists generally of duplicating or triplicating computations and voting on these copies [43]. By reducing or eliminating fault-tolerance, significant workload reductions are possible. We can do this and still retain system safety if we adapt the amount of fault-tolerance to the current state of the controlled plant [10].

Yet another approach is applicable whenever the workload consists of anytime or IRIS (Increased Reward with Increased Service) tasks [61]. Such tasks can be terminated prematurely at the cost of reduced precision. The penalty for a reduced workload burden is a reduction in the quality of the computational output. Depending on the state of the controlled plant and its dynamics, this may or may not be acceptable. See [62] for an example (non-thermal-related) where a controller in a cyber-physical system sends requests in the form of (δ, ϵ); here, δ is the delay and ϵ the quality of output.

Alternatively, we may use *demand shapers* [63]. A demand shaper adjusts the flow of tasks. Denote the demand shaper by the function $\sigma(\cdot)$. Then, if the arrival rate of a given task is bounded by $d(\tau)$ (meaning that this is the maximum number of its iterations that can arrive in an interval of duration τ), then the output of the demand shaper is a shaped arrival rate, denoted by $d_s(\tau)$,

$$d_s(\tau) = d \otimes \sigma = \inf_{0 \leq \xi \leq \tau} \{d(\tau - \xi) + \sigma(\xi)\}.$$

It is possible to show that the shaper ensures that the number of jobs (i.e., iterations) of this task arriving in any interval of duration τ is upper bounded by $\sigma(\tau)$. Furthermore, we can obtain an upper bound on the amount of time by which the shaper can delay a job; we can also set the shaper in such a way as to reduce the peak temperature of the chip (see [63] for details). We can therefore set the shaper to ensure that no task misses its deadline while its arrival to the system ready queue is adjusted to the extent possible.

Thus far, we have concentrated on uniprocessors running sets of independent tasks. The scheduling approach can be extended to cover multiprocessors. One approach is to carry out a two-phase algorithm: in the first phase, we allocate tasks to processors; in the second, we run a uniprocessor scheduling algorithm (e.g., RM or EDF), to decide when the tasks should be run. See [61], [8] for details. (Note, however, that analyzing multicore activity is a very difficult task owing to the complex interference patterns between one core and another, in their competition for common resources. Indeed, this difficulty is such that in the most critical applications, users are sometimes forced to work with single-core systems by using just one core in a multicore processor and turning off or idling the rest [54].)

Scheduling of task graphs, as opposed to independent tasks, is much more difficult. One approach is to assign virtual deadlines and virtual release times to each task; set the virtual release time of a task to be greater than the virtual deadline of each of its precedent tasks [64]. Another approach is to set the problem up as an integer linear programming and solve it [65].

IX. Discussion

With the increasing thermal density of today's semiconductor devices and the emergence of complex cyber-physical applications, has come the need to prevent device overheating

while still delivering adequate levels of service to the application. There is a large number of thermal-aware management schemes in the literature. These all derive from just a handful of relationships: between frequency and voltage on the one hand and energy consumption on the other, between energy dissipation and temperature, the exponential acceleration of many failure processes with temperature, and the workload demands of the cyber-physical application, as well as the adaptation of such workload to the state of that application. In this paper, we have presented the principal ways in which these reliationships have been exploited in order to reduce thermal stress while satisfying the computational demands of the cyber-physical application.

ACKNOWLEDGEMENT

The authors thank the reviewers for their careful reading of the manuscript and their helpful comments. This work was partially supported by the National Science Foundation under CNS-1329831.

REFERENCES

[1] Ram Viswanath, Vijay Wakharkar, Abhay Watwe, Vassou Lebonheur, et al. Thermal performance challenges from silicon to systems. 2000.

[2] Israel Koren and CM Krishna. Temperature-aware computing. *Sustainable Computing: Informatics and Systems*, 1(1):46–56, 2011.

[3] Hafiz Fahad Sheikh, Ishfaq Ahmad, Zhe Wang, and Sanjay Ranka. An overview and classification of thermal-aware scheduling techniques for multi-core processing systems. *Sustainable Computing: Informatics and Systems*, 2(3):151–169, 2012.

[4] Jayanth Srinivasan, Sarita V Adve, Pradip Bose, and Jude A Rivers. The impact of technology scaling on lifetime reliability. In *Dependable Systems and Networks, 2004 International Conference on*, pages 177–186. IEEE, 2004.

[5] RL De Orio, Hajdin Ceric, and Siegfried Selberherr. Physically based models of electromigration: From blacks equation to modern tcad models. *Microelectronics Reliability*, 50(6):775–789, 2010.

[6] Cheng Zhuo, Dennis Sylvester, and David Blaauw. Process variation and temperature-aware reliability management. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 580–585. European Design and Automation Association, 2010.

[7] Rehan Ahmed, Parameswaran Ramanathan, and Kewal K Saluja. On thermal utilization of periodic task sets in uni-processor systems. In *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 267–276. IEEE, 2013.

[8] Jane W.S. Liu. *Real-Time Systems*. Wiley, 2000.

[9] Anup Das, Akash Kumar, and Bharadwaj Veeravalli. Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia mpsocs. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 102. European Design and Automation Association, 2014.

[10] C Mani Krishna. Ameliorating thermally accelerated aging with state-based application of fault-tolerance in cyber-physical computers. *IEEE Transactions on Reliability*, 64(1):4–14, 2015.

[11] Pradeep M Hettiarachchi, Nathan Fisher, Masud Ahmed, Le Yi Wang, Shinan Wang, and Weisong Shi. A design and analysis framework for thermal-resilient hard real-time systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(5s):146, 2014.

[12] John F. Meyer. On evaluating the performability of degradable computing systems. *IEEE Transactions on computers*, 100(8):720–731, 1980.

[13] Bernie Siegal. An introduction to diode thermal measurements. *Thermal Engineering Associates, Inc., Santa Clara, CA. USA*, 2009.

[14] Chan-Kyung Kim, Jae-Goo Lee, Young-Hyun Jun, Chil-Gee Lee, and Bai-Sun Kong. Cmos temperature sensor with ring oscillator for mobile dram self-refresh control. *Microelectronics Journal*, 38(10):1042–1049, 2007.

[15] Spandana Remarsu and Sandip Kundu. On process variation tolerant low cost thermal sensor design in 32nm cmos technology. In *Proceedings of the 19th ACM Great Lakes symposium on VLSI*, pages 487–492. ACM, 2009.

[16] Chunhua Yao, Kewal K Saluja, and Parmesh Ramanathan. Calibrating on-chip thermal sensors in integrated circuits: A design-for-calibration approach. *Journal of Electronic Testing*, 27(6):711–721, 2011.

[17] Shirley Browne, Jack Dongarra, Nathan Garner, Kevin London, and Philip Mucci. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Supercomputing, ACM/IEEE 2000 Conference*, pages 42–42. IEEE, 2000.

[18] Xiao Zhang, Sandhya Dwarkadas, Girts Folkmanis, and Kai Shen. Processor hardware counter statistics as a first-class system resource. In *HotOS*, 2007.

[19] Sung Woo Chung and Kevin Skadron. Using on-chip event counters for high-resolution, real-time temperature measurement. In *Thermal and Thermomechanical Proceedings 10th Intersociety Conference on Phenomena in Electronics Systems, 2006. ITHERM 2006.*, pages 114–120. IEEE, 2006.

[20] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 93. IEEE Computer Society, 2003.

[21] Rance Rodrigues, Arunachalam Annamalai, Israel Koren, and Sandip Kundu. A study on the use of performance counters to estimate power in microprocessors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 60(12):882–886, 2013.

[22] Mayank Chhablani, Israel Koren, and C.M. Krishna. Online inertia-based temperature estimation for reliability enhancement. *Journal of Low Power Electronics*, 12(3), 2016.

[23] Charu C Aggarwal. *Data mining: the textbook*. Springer, 2015.

[24] Buyoung Yun, Kang G Shin, and Shige Wang. Thermal-aware scheduling of critical applications using job migration and power-gating on multi-core chips. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1083–1090. IEEE, 2011.

[25] Yang Ge, Parth Malani, and Qinru Qiu. Distributed task migration for thermal management in many-core systems. In *Proceedings of the 47th Design Automation Conference*, pages 579–584. ACM, 2010.

[26] Wei Huang, Shougata Ghosh, Sivakumar Velusamy, Karthik Sankaranarayanan, Kevin Skadron, and Mircea R Stan. Hotspot: A compact thermal modeling methodology for early-stage vlsi design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(5):501–513, 2006.

[27] Yongkui Han. *Temperature aware techniques for design, simulation and measurement in microprocessors*. ProQuest, 2007.

[28] Karl J Åström and Björn Wittenmark. *Computer-controlled systems: theory and design*. Courier Corporation, 2013.

[29] Yongkui Han, Israel Koren, and C Mani Krishna. Tilts: A fast architectural-level transient thermal simulation method. *Journal of Low Power Electronics*, 3(1):13–21, 2007.

[30] Lars Schor, Iuliana Bacivarov, Hoeseok Yang, and Lothar Thiele. Worst-case temperature guarantees for real-time applications on multi-core systems. In *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, pages 87–96. IEEE, 2012.

[31] Lars Schor, Hoeseok Yang, Iuliana Bacivarov, and Lothar Thiele. Thermal-aware task assignment for real-time applications on multi-core systems. In *Formal Methods for Components and Objects*, pages 294–313. Springer, 2011.

[32] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[33] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–480. ACM, 2009.

[34] Shikang Xu, Israel Koren, and CM Krishna. Improving processor lifespan and energy consumption using dvfs based on ilp monitoring. In *Green Computing Conference and Sustainable Computing Conference (IGSC), 2015 Sixth International*, pages 1–6. IEEE, 2015.

[35] Rehan Ahmed, Parameswaran Ramanathan, and Kewal K Saluja. Necessary and sufficient conditions for thermal schedulability of periodic real-time tasks under fluid scheduling model. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(3):49, 2016.

[36] Yongpan Liu, Robert P Dick, Li Shang, and Huazhong Yang. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *Proceedings of the conference on Design, automation and test in Europe*, pages 1526–1531. EDA Consortium, 2007.

[37] Rehan Ahmed, Parameswaran Ramanathan, and Kewal K Saluja. Temperature minimization using power redistribution in embedded systems. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 264–269. IEEE, 2014.

[38] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[39] Huang Huang, Vivek Chaturvedi, Gang Quan, Jeffrey Fan, and Meikang Qiu. Throughput maximization for periodic real-time systems under the maximal temperature constraint. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(2s):70, 2014.

[40] Rance Rodrigues, Arunachalam Annamalai, Israel Koren, and Sandip Kundu. Improving performance per watt of asymmetric multi-core processors via online program phase classification and adaptive core morphing. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(1):5, 2013.

[41] Huaping Wang, Israel Koren, and C Mani Krishna. Utilization-based resource partitioning for power-performance efficiency in smt processors. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1150–1163, 2011.

[42] Huaping Wang, Israel Koren, and C Mani Krishna. Runtime architecture adaptation for energy management in embedded real-time systems. In *Green Computing Conference (IGCC), 2012 International*, pages 1–9. IEEE, 2012.

[43] Israel Koren and C Mani Krishna. *Fault-tolerant systems*. Morgan Kaufmann, 2010.

[44] Mohamed Gomaa, Michael D Powell, and TN Vijaykumar. Heat-and-run: leveraging smt and cmp to manage power density through the operating system. In *ACM SIGARCH Computer Architecture News*, volume 32, pages 260–270. ACM, 2004.

[45] Ting-Hao Tsai and Ya-Shu Chen. Thermal-throttling server: A thermal-aware real-time task scheduling framework for three-dimensional multicore chips. *Journal of Systems and Software*, 112:11–25, 2016.

[46] Albert MK Cheng. *Real-time systems: scheduling, analysis, and verification*. John Wiley & Sons, 2003.

[47] Buyoung Yun, Kang G Shin, and Shige Wang. Predicting thermal behavior for temperature management in time-critical multicore systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, pages 185–194. IEEE, 2013.

[48] Bagher Salami, Mohammadreza Baharani, and Hamid Noori. Proactive task migration with a self-adjusting migration threshold for dynamic thermal management of multi-core processors. *The Journal of Supercomputing*, 68(3):1068–1087, 2014.

[49] Zao Liu, Sheldon X-D Tan, Xin Huang, and Hai Wang. Task migrations for distributed thermal management considering transient effects. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(2):397–401, 2015.

[50] Farid Golnaraghi and Benjamin C Kuo. *Automatic Control Systems*. Academic Press, 2009.

[51] Katsuhiko Ogata. *Modern Control Engineering*. Pearson, 2009.

[52] Yong Fu, Nicholas Kottenstette, Yingming Chen, Chenyang Lu, Xenofon D Koutsoukos, and Hongan Wang. Feedback thermal control for real-time systems. In *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 111–120. IEEE, 2010.

[53] Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)*, 43(4):35, 2011.

[54] Bryan C Ward. Relaxing resource-sharing constraints for improved hardware management and schedulability. In *Real-Time Systems Symposium, 2015 IEEE*, pages 153–164. IEEE, 2015.

[55] Giorgio C Buttazzo, Marko Bertogna, and Gang Yao. Limited preemptive scheduling for real-time systems. a survey. *IEEE Transactions on Industrial Informatics*, 9(1):3–15, 2013.

[56] Alan Burns and Robert Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, 2013.

[57] Sanjoy Baruah. The federated scheduling of constrained-deadline sporadic dag task systems. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 1323–1328. EDA Consortium, 2015.

[58] Zhishan Guo and Sanjoy K Baruah. Uniprocessor edf scheduling of avr task systems. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, pages 159–168. ACM, 2015.

[59] Junsung Kim, Hyoseung Kim, Karthik Lakshmanan, and Ragunathan Raj Rajkumar. Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, pages 31–40. ACM, 2013.

[60] Shengquan Wang, Youngwoo Ahn, and Riccardo Bettati. Schedulability analysis in hard real-time systems under thermal constraints. *Real-Time Systems*, 46(2):160–188, 2010.

[61] C.M. Krishna and Kang G. Shin. *Real-Time Systems*. McGraw-Hill, 1996.

[62] Yash Vardhan Pant, Houssam Abbas, Kartik Mohta, Truong X Nghiem, Joseph Devietti, and Rahul Mangharam. Co-design of anytime computation and robust control. In *Real-Time Systems Symposium, 2015 IEEE*, pages 43–52. IEEE, 2015.

[63] Pratyush Kumar and Lothar Thiele. Cool shapers: shaping real-time tasks for improved thermal guarantees. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 468–473. IEEE, 2011.

[64] Riccardo Bettati and JW-S Liu. End-to-end scheduling to meet deadlines in distributed systems. In *Distributed Computing Systems, 1992., Proceedings of the 12th International Conference on*, pages 452–459. IEEE, 1992.

[65] Ayse K Coskun, Tajana S Rosing, Keith A Whisnant, and Kenny C Gross. Static and dynamic temperature-aware scheduling for multiprocessor socs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(9):1127–1140, 2008.