# The Design and Analysis of Thermal-Resilient Hard-Real-Time Systems

Pradeep M. Hettiarachchi[1], Nathan Fisher[1], Masud Ahmed[1], Le Yi Wang[2], Shinan Wang[1], and Weisong Shi[1]

[1]Department of Computer Science
[2]Department of Electrical and Computer Engineering
Wayne State University
{pradeepmh, fishern, masud, lywang, shinan, weisong}@wayne.edu

***Abstract*—We address the challenge of designing predictable real-time systems in an unpredictable thermal environment where environmental temperature may dynamically change (e.g., implantable medical devices). Towards this challenge, we propose a control-theoretic design methodology which permits a system designer to specify a set of hard-real-time performance modes under which the system may operate. The system automatically adjusts the real-time performance mode based on the external thermal stress. We show (via analysis, simulations, and a hardware testbed implementation) that our control-design framework is stable and control performance is equivalent to previous real-time thermal approaches, even under dynamic temperature changes. A crucial and novel advantage of our framework over previous real-time control is the ability to guarantee hard deadlines even under transitions between modes. Furthermore, our system design permits the calculation of a new metric called *thermal resiliency* which characterizes the maximum external thermal stress that any hard-real-time performance mode can withstand. Thus, our design framework and analysis may be classified as a *thermal stress analysis* for real-time systems.**

***Index Terms*—thermal resiliency; multi-mode system; thermal-aware system; thermal-aware periodic resource;**

## I. Introduction

Modern computer-controlled systems are often deployed in dynamic and unpredictable thermal operating environments. From the hardware-design perspective, material scientists and computer engineers use rigorous *thermal-stress analysis* techniques (e.g., see [1]) to determine how the underlying physical hardware will withstand applied internal and external thermodynamic forces. Unfortunately, equivalent analysis does not exist for determining the effects of (unpredictable) thermal stress on the performance of the systems software. While hardware capabilities such as dynamic power management (DPM) permit a computing system to reduce its power dissipation at run-time, many embedded systems have real-time constraints which may be adversely affected by unexpected changes in processor speed.

As an example of an embedded system where thermal-stress analysis is essential, consider microprocessors found in implantable medical devices (IMDs). IMDs are increasingly being used to treat various diseases and medical conditions (e.g., pacemakers for heart disease or neural implants to restore hearing/vision). However, recent studies [2], [3] have shown that the heat dissipated from IMDs due to the microprocessor activity is non-negligible. Thus, designing IMDs with minimum thermal dissipation is critical as medical research has shown that a temperature increase of even $1^{\circ}C$ can have long-term effect on tissue [4] and, in the extreme, death may even result from excessive tissue heating [5]. Complicating the safe thermal design of IMDs, body temperature naturally fluctuates over time and varies depending on location [6]. An IMD designer must balance (under temperature fluctuations) the real-time computational requirements of the device with the non-harmful thermal operating limits. In the presence of an increased surrounding temperature, an IMD will have to reduce its computational load to prevent tissue damage due to heat[1]. However, as the correct and safe functioning of the IMD is an absolute requirement, the system designer requires techniques to formally verify the effect of different body temperatures on the correct operation of the IMD. Similarly, as a less safety-critical example, consider how the quality of audio/video decoding may degrade in a hand-held device as the system reacts to increases in temperature by reducing computational processing (e.g., via instruction fetch toggling). Ideally, a system designer would like to determine how much the performance will degrade under different thermal operating conditions.

Unfortunately, no current formal real-time design and analysis framework fully addresses the above setting. Recently-proposed control-theoretic frameworks exist for regulating processor temperature for soft-real-time systems (i.e., systems where jobs are permitted to "occasionally" miss computational deadlines) in an unpredictable thermal environment [8], [9]. While their results successfully show that it is possible to obtain stable and responsive thermal behavior and system utilization control, a system designer cannot use their approaches to *a priori* determine the amount of system-performance degradation due to changes in the thermal environment. Instead, the level of degradation can only be indirectly inferred via simulations of the system for different operating conditions. Furthermore, hard timing guarantees cannot be made in these frameworks. Techniques also already exist for permitting a

[1]As IMD microprocessors typically do not have DVS capabilities, an IMD may have to reduce non-essential tasks such as communication with other nodes in a body-area network [7].

trade-off between real-time QoS and processing resources (e.g., the *QoS-based resource allocation model* (QRAM) [10]); however, while such techniques may guarantee real-time deadlines under a fixed level of resources, they cannot guarantee deadlines when a system must dynamically switch between real-time modes (due to the uncompleted execution remaining at mode transitions). Furthermore, none of these previously-proposed techniques can be used to obtain a precise, formal quantification of the thermal stress that the system can withstand.

In this paper, we address the challenge of determining the real-time guarantees in the presence of unpredictable dynamic environmental conditions. Towards this goal, we propose a framework and mechanisms for thermal-stress analysis in real-time systems. Our objective is *to develop techniques that permit a system designer to specify, a priori, a precise quantification of the hard-real-time performance degradation due to external thermal events, via a new system design metric called* **real-time thermal resiliency**. Informally, real-time thermal resiliency is a prediction of the maximum external operating temperature at which a specified *real-time performance mode* (e.g., quality-of-service) may be guaranteed in the system steady-state (i.e., a time at which system properties have converged and do not change). To illustrate, consider a system with $q$ different (system designer-defined) hard-real-time performance modes $M_0, M_1, \ldots, M_q$ where modes are ordered in increasing levels of real-time performance with $M_q$ guaranteeing the highest level and $M_0$ the lowest. The real-time thermal resiliency of any mode $M_i$, denoted as $\Lambda(M_i, \mathcal{T}_{\text{ref}})$, is the predicted maximum external operating temperature for which the system will continue to operate (in the steady state) at performance mode $M_i$ or higher and maintain a CPU reference temperature of $\mathcal{T}_{\text{ref}}$. Furthermore, if the external temperature exceeds $\Lambda(M_i, \mathcal{T}_{\text{ref}})$, then the system should automatically degrade to the next lowest performance mode $M_{i-1}$. The capability to define (at system-design time) thermal-resilient, real-time performance modes allows the system designer to specify how a system will gracefully and predictably degrade under external thermal stress; furthermore, the ability to accurately determine the real-time thermal resiliency of a performance mode provides a real-time system designer with a thermal-stress analysis framework analogous to stress analysis techniques in physical sciences and engineering. In the IMD example above, the thermal-resiliency function $\Lambda$ may be used to determine (at design time) the body-temperature that a given set of tasks may safely operate at without doing damage to surrounding tissue.

§**Organization.** This paper presents a methodology for designing and analyzing thermal-resilient hard-real-time systems. Section II presents a high-level overview of our methodology and gives more detail on the contributions of this paper. Section III presents a brief review of previous work on thermal-aware (real-time and non-real-time) computer systems. Section IV presents the hardware, real-time, and thermal models used throughout the paper. Section V details the design of our thermal-resilient controller. Section VI derives thermal-resiliency function $\Lambda$ for control system. Section VII describes the results of our comparison with previous control systems via simulation and implementation upon testbed hardware. Our methodology provides formal system guarantees which require formal derivations and proofs. In the interest of space, we have deferred all formal proofs and derivations to the appendix of an extended version of this paper [11].

## II. Methodology Overview

We now describe at a high level the major steps of our thermal-resilient design and analysis methodology.

1) **System Hardware Specification:** In the first step, the system designer must specify the processing and DPM capabilities of the system. Throughout this paper, we will be illustrating and validating our methodology upon an Intel Pentium IV 3.0 GHz single-core processor testbed. To match the rudimentary DPM capabilities often present in embedded processors, our testbed possesses the ability to only modulate the power modes of the system between active and inactive states. Section IV-A gives more detail on the hardware model and our testbed implementation details.
2) **System Software Specification:** The system designer must specify the set of valid software modes $M_0, M_1, \ldots, M_q$ for the system. In Section IV-B, we discuss using the sporadic task model [12] as a model for real-time workload of each software mode.
3) **Real-Time Mode Resource Allocation:** After the HW/SW specification steps, the designer must determine the minimum resource allocation under which the multi-mode system is schedulable. We discuss in Section IV-B how recent techniques for schedulability analysis of hard-real-time systems where both the hardware and software change modes may be used in allocating sufficient processing time to each mode.
4) **Power/Thermal Model Evaluation:** Given the processing platform, we need an accurate power model in order to derive formal guarantees on the thermal resiliency of the system. Due to the duality between electrical and thermal circuits, we model the thermodynamics of our processing system using the resistance/capacitance (RC) circuits. We use *system identification* (SI) to identify the system parameters and evaluate the efficacy of our power-model choice. Due to space constraints, the details on the derived parameters for our hardware testbed are in the appendix of the extended version of our paper [11].
5) **Control System Design:** We design a control structure based on optimal control theory. In this process, we use the SI parameters (determined in the previous step) to design the feedback gain parameters. We present details on our controller design in Section V.
6) **System Simulation:** We build a system simulator which implements the real-time scheduling algorithm and control algorithm and simulates the real-time and thermal

behavior of the system based on the resource allocations and power model derived in Steps 3 and 4. The details of our simulator are provided in Section VII.

7) **Thermal-Resiliency Function Calculation:** Given the real-time mode resource allocation, power model, controller, and simulator observations obtained from Steps 3, 4, 5, and 6 we can obtain a quantification of the thermal-resiliency function $\Lambda$. We give details on the derivation of this function in Section VI.
8) **System Validation:** We finally validate our system simulator and thermal-resiliency calculations in Section VII by comparing directly with observations from our hardware testbed. Our comparison shows that the system simulator closely models the actual testbed behavior. Furthermore, we validate that our predicted thermal-resiliency $\Lambda$ function is accurate by observing that it closely tracks the actual hardware testbed behavior.

While most of the steps above are standard practice in control system design, we would like to emphasize that our ability to ensure the hard-real-time schedulability of each mode in Step 3 and obtain *a priori* guarantees on thermal resiliency in Step 7 distinguishes our approach from previous thermal control for real-time systems.

## III. Related Work

In this section, we give a brief, high-level overview of previous research in both general (non-real-time), thermal-aware system design and real-time-specific thermal-aware design. For non-real-time systems, Brooks and Martonosi [13] investigated major components of any dynamic thermal management scheme and suggested policies and mechanisms for implementing dynamic thermal management for current and future high-end CPUs. They evaluated the benefits of using dynamic thermal management to reduce the cooling system costs of CPUs and developed an architectural-level power modeling tool called Wattch. For the micro-architecture level of thermal modeling, Skadron et al. [14] proposed a compact, dynamic, and portable thermal model and a tool called *HotSpot* for use at the architecture level for micro-architectures.

For real-time systems in the online setting, Bansal and Pruhs [15] explored algorithms for minimizing both peak-temperature and energy efficiency for online jobs with deadline constraints. In the off-line setting, previous work on scheduling under thermal constraints has followed two main approaches: reactive and proactive schedulers. In a reactive scheduler, the processor speed is reduced in response to a thermal trigger. Wang et al. [16] studied schedulability analysis under the reactive setting. In the proactive setting, the speed schedule for the processor is determined at design time. Chen et al. [17] addressed proactive scheduling for the periodic task model. Quan and Zhang [18] consider feasibility analysis of leakage-aware periodic tasks under temperature constraints. However, previous work on both settings assumed either simple task models or the existence of "ideal" processor speeds. Our proposed control framework may be considered a proactive scheduler; however, we attempt to remove some ideal assumptions by working with only two power modes and the more general sporadic task model. Also, we consider the ambient temperature changes and analyze the effects on the task system due to its variation. Recent dynamic temperature management strategies also exist for multiprocessor real-time systems [19]–[21]; however, most of these focus upon static speed-assignment approaches and not a proactive schedule. Thermal analysis has also been studied in the context of web servers [22], but hard deadlines are not guaranteed. As mentioned in the introduction, work by Y. Fu et al. [8] and X. Fu et al. [9] address handling unpredictable thermal events; however, the results do not provide any *a priori* guarantees that may be used to equate real-time performance and thermal resiliency.

## IV. Models

### A. System Hardware Model and Testbed

For this paper, we consider a single processor system with rudimentary DPM capabilities of only *active* and *inactive* power modes. At any time $t > 0$, we denote the instantaneous CPU power as $\mathcal{P}_{\text{cpu}}(t)$. The processor dissipates thermal power at a constant rate $\mathcal{P}_{\text{cpu}}(t) = \mathcal{P}_{\text{act}}$ in the active mode and $\mathcal{P}_{\text{cpu}}(t) = \mathcal{P}_{\text{inc}}$ in the inactive mode. Also, we assume that processor consumes $e_{\text{act}}$ amount of energy to activate from inactive mode and $e_{\text{inc}}$ amount of energy to deactivate from the active mode. Even though the processor may be minimally active while in the low-power state, we will assume (as a pessimistic assumption for the purpose of schedulability analysis) that the processor is unavailable for task execution during this interval. If the aforementioned assumption does not hold, the system will behave "better" than the analysis and our results will continue to be valid. We believe this model of active/inactive modes is a very general model, applicable to a large number of available embedded processors with rudimentary DPM capabilities. For ideal processors with continuous power modes, $\mathcal{P}_{\text{cpu}}(t)$ may be selected from the range $[0, \mathcal{P}_{\text{act}}]$.

Our control system for the active/inactive processor will enforce strict periodic mode changes. For this purpose, we employ a recently proposed *thermal-aware periodic resource* [23] model, which is an extension of the well-known periodic resource model proposed by Shin and Lee [24] for compositional real-time systems. In the thermal-aware periodic resource model, the processing resource is characterized with a two-tuple $(\Pi, \Theta)$. The parameter $\Pi$ is called the *resource period* and $\Theta$ is called the *resource capacity*. We will assume that $\Pi$ is a non-negative integer (likely subject to the system tick granularity). The interpretation is that processor will be active for $\Theta$ amount of time at the beginning of each successive $\Pi$-length intervals. The ratio $\Theta/\Pi$ is called the *resource bandwidth*. Within each processor allocation, an arbitrary uniprocessor scheduling algorithm (e.g., EDF or RM) may be employed to schedule the underlying task system (see next subsection). See Figure 1 for an illustration of the

thermal-aware periodic resource.

As a case study of our methodology, we have built a hardware testbed using an Intel Pentium IV 3.0 GHz single core processor running a modified Linux kernel (2.6.33.7.2-rt30 PREEMPT_RT). We have developed device drivers to activate the CPU modulation from the user-space using Model Specific Registers (MSR) of the processor to create a high and low frequency modulation for active/inactive power states. For obtaining the system temperature, we follow the official procedure given by Intel [25] to install a thermal sensor to measure the die temperature with best possible accuracy. A Phidgets four-port thermal sensor with ambient temperature sensor was used to measure the air and environment temperature. A detailed description on the testbed is given in the extended version of this paper [11].

### B. System Software Model

In the introduction, we proposed a system model of *real-time performance modes* $M_1, \ldots, M_q$. For the purpose of this paper, we will assume each performance mode $M_i$ is characterized by a *sporadic task system*[2] [12] with $n_i$ tasks and the resource capacity $\Theta^{(i)}$. That is, $M_i = \left(\left\{\tau_1^{(i)}, \tau_2^{(i)}, \ldots, \tau_{n_i}^{(i)}\right\}, \Theta^{(i)}\right)$ where each $\tau_j^{(i)} \in M_i$ is a sporadic task characterized by a three-tuple $(e_j^{(i)}, d_j^{(i)}, p_j^{(i)})$ and $\Theta^{(i)}$ is the minimum capacity required to meet the deadlines of the tasks of $M_i$. (Note that we are abusing notation by allowing $M_i$ to represent the set of tasks and the two-tuple of the mode's task system and required resource capacity.) In this three-tuple representation for a task, $e_j^{(i)}$ is the *worst-case execution requirement*, $d_j^{(i)}$ is the *relative deadline*, and $p_j^{(i)}$ is the *minimum inter-arrival separation parameter* (historically called the "period"). A sporadic task $\tau_j^{(i)}$ may produce a (potentially infinite) sequence of jobs, where each job has an execution requirement of $e_j^{(i)}$ time units and must complete $d_j^{(i)}$ time units after its arrival. The first job of $\tau_j^{(i)}$ may arrive at any time after system-start time; however, successive jobs of $\tau_j^{(i)}$ must arrive at least $p_j^{(i)}$ time units apart. For this paper, we assume that the resource period $\Pi$ is identical in all modes. For mode $M_i$, a resource capacity of $\Theta^{(i)}$ is provided every resource period. Figure 1 illustrates the processing-time allocation in two different modes.

We will assume that there is an ordering of real-time performance modes based on their "computational requirements" to meet all of a mode's deadlines. The relation $M_i \succeq M_j$ indicates that $M_i$ is more computationally intensive than $M_j$. For notational convenience, we will assume that mode $M_0$ represents the mode where with no tasks and $\Theta^{(0)}$ equal to zero. Furthermore, for this paper, we assume that the modes are well-ordered and have been indexed in increasing order of computational requirements; i.e., $M_0 \preceq M_1 \preceq M_2 \preceq \ldots \preceq M_q$. While there are many possible ways to define the $\preceq$ relation, the only ordering required from the perspective of our thermal control is that $M_i \preceq M_j$, if and only if, $\Theta^{(i)} \leq \Theta^{(j)}$; i.e., to reduce the temperature of the system, we need to decrease the processing-time allocation.

[2]Note, we will be assuming the sporadic task model throughout our objectives, but the results could be extended to other task models without much change.

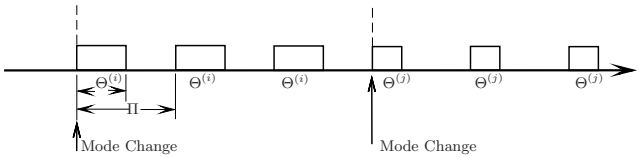


Fig. 1: The sampling and mode change in our thermal control system. The blocks indicate time periods during with the processor is active under the thermal-aware periodic resource model. Sporadic tasks are scheduled within the activation blocks.

Our model does not require any particular mode-change semantics to be adopted. Some potential options for dealing with incompletely-executed jobs upon a mode change are: (*i*) aborting any incomplete jobs; (*ii*) delaying the release of jobs in the new mode until all jobs of the old mode have completed; and (*iii*) allowing jobs of the new mode to be released, as soon as legally allowable, while jobs of the old mode are still active. For the purposes of our hardware testbed and simulations (Section VII), we assume option (*iii*).

The scheduling of real-time performance mode $M_i$ upon the thermal-aware periodic resource may be done by any uniprocessor real-time scheduling algorithm (e.g., earliest-deadline-first or rate-monotonic [26]). However, $\Theta^{(i)}$ must be sufficiently large for the scheduling algorithm to correctly schedule all jobs of the task set of $M_i$ (i.e., $\{\tau_1^{(i)}, \tau_2^{(i)}, \ldots, \tau_{n_i}^{(i)}\}$) and (potentially) any jobs from the previous mode that have not completed by the mode change. To obtain a proper resource allocation, $\Theta^{(i)}$, for each mode, we use our recently-developed hard-real-time schedulability test (for EDF scheduling under hardware/software mode changes in the periodic resource model) to search for a safe value of $\Theta^{(i)}$ for each mode [27] to ensure that deadlines are always met.

### C. Power/Thermal Model

We use the duality principle in electrical and thermal circuits to describe the dynamics of the power dissipating source using electrical resistance/capacitance (RC) circuits. Figure 2 shows the basic equivalent circuit for the CPU and its surrounding environment. We assume that total dissipated power of the CPU $\mathcal{P}_{\text{cpu}}$ is equal to the sum of the power due to dynamic current $\mathcal{P}_{\text{cpu}}^{\text{d}}$ and power due to leakage current $\mathcal{P}_{\text{cpu}}^{\text{l}}$. Furthermore, we assume that the temperature-dependant leakage power may be closely approximated by a linear function of CPU temperature [28].

Let $V_{\text{cpu}}(t), V_{\text{env}}(t)$, and $V_{\text{air}}(t)$ represent the equivalent voltages for equivalent temperatures of the CPU, environment, and air (room) respectively. Let $\mathcal{T}_{\text{cpu}}$ be the instantaneous relative temperature of the CPU with respect to the immediate environment (e.g., CPU casing), $\mathcal{T}_{\text{env}}$ be the relative temperature of the immediate environment with respect to the room air temperature, and $\mathcal{T}_{\text{air}}$ be the (absolute) room air temperature.

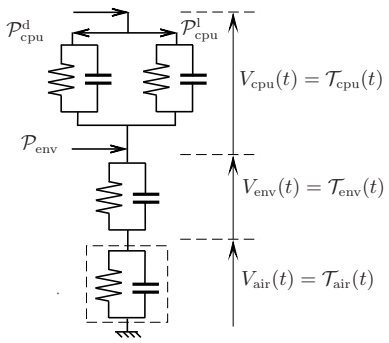


Fig. 2: The basic equivalent circuit for a working CPU and its working environment

For example, if $\mathcal{T}_{\mathrm{air}}$ is $20°C$, $\mathcal{T}_{\mathrm{env}}$ is $10°C$, and $\mathcal{T}_{\mathrm{cpu}}$ is $15°C$, then the absolute temperature of the CPU is $45°C$.

Let $\mathcal{P}^{\mathrm{d}}_{\mathrm{cpu}}(t)$, $\mathcal{P}^{\mathrm{l}}_{\mathrm{cpu}}(t)$, and $\mathcal{P}_{\mathrm{env}}(t)$ represent, respectively, the dynamic CPU, leakage CPU, and environment power dissipation. Let $R^{\mathrm{d}}_{\mathrm{cpu}}$, $R^{\mathrm{l}}_{\mathrm{cpu}}$, $R_{\mathrm{env}}$, $C^{\mathrm{d}}_{\mathrm{cpu}}$, $C^{\mathrm{l}}_{\mathrm{cpu}}$, and $C_{\mathrm{env}}$ represent the dynamic and leakage thermal resistance, environment resistance, CPU dynamic and leakage capacitance, and environment capacitance. Finally, let $\sigma_1 \stackrel{\mathrm{def}}{=} \frac{1}{C^{\mathrm{d}}_{\mathrm{cpu}}+C^{\mathrm{l}}_{\mathrm{cpu}}}$ and $k_T$ and $k_C$ represent processor-dependent constants used in approximating the temperature-dependant leakage current. Applying Kirchhoff's circuit laws, we get the following equations for $\mathcal{T}_{\mathrm{cpu}}(t)$,

$$\frac{\mathcal{T}_{\mathrm{cpu}}(t)}{R^{\mathrm{d}}_{\mathrm{cpu}}} + C^{\mathrm{d}}_{\mathrm{cpu}}\frac{d}{dt}\mathcal{T}_{\mathrm{cpu}}(t) = \mathcal{P}^{\mathrm{d}}_{\mathrm{cpu}}(t) \quad (1)$$

$$\begin{aligned}\frac{\mathcal{T}_{\mathrm{cpu}}(t)}{R^{\mathrm{l}}_{\mathrm{cpu}}} + C^{\mathrm{l}}_{\mathrm{cpu}}\frac{d}{dt}\mathcal{T}_{\mathrm{cpu}}(t) &= \mathcal{P}^{\mathrm{l}}_{\mathrm{cpu}}(t) \quad (2)\\ &= k_T\big(\mathcal{T}_{\mathrm{cpu}}(t) + \mathcal{T}_{\mathrm{env}}(t)\big) + k_C.\end{aligned}$$

Solving (2) for $\frac{d}{dt}\mathcal{T}_{\mathrm{cpu}}(t)$,

$$\begin{aligned}\frac{d}{dt}\mathcal{T}_{\mathrm{cpu}}(t) &= \sigma_1\Big(k_T - \frac{1}{R^{\mathrm{l}}_{\mathrm{cpu}}} - \frac{1}{R^{\mathrm{d}}_{\mathrm{cpu}}}\Big)\mathcal{T}_{\mathrm{cpu}}(t)\\ &+ k_T\sigma_1\mathcal{T}_{\mathrm{env}}(t) + \sigma_1\mathcal{P}^{\mathrm{d}}_{\mathrm{cpu}}(t) + \sigma_1 k_C. \quad (3)\end{aligned}$$

We obtain the following equation for $\mathcal{T}_{\mathrm{env}}(t)$,

$$\begin{aligned}\frac{\mathcal{T}_{\mathrm{env}}(t)}{R_{\mathrm{env}}} + C_{\mathrm{env}}\frac{d}{dt}\mathcal{T}_{\mathrm{env}}(t) &= \mathcal{P}_{\mathrm{cpu}}(t) + \mathcal{P}_{\mathrm{env}}(t), \quad (4)\\ &= \mathcal{P}^{\mathrm{d}}_{\mathrm{cpu}}(t) + \mathcal{P}^{\mathrm{l}}_{\mathrm{cpu}}(t) + \mathcal{P}_{\mathrm{env}}(t).\end{aligned}$$

Solving (4) for $\frac{d}{dt}\mathcal{T}_{\mathrm{env}}(t)$,

$$\begin{aligned}\frac{d}{dt}\mathcal{T}_{\mathrm{env}}(t) &= \frac{k_T}{C_{\mathrm{env}}}\mathcal{T}_{\mathrm{cpu}}(t) + \frac{1}{C_{\mathrm{env}}}\mathcal{P}^{\mathrm{d}}_{\mathrm{cpu}}(t) + \frac{1}{C_{\mathrm{env}}}\mathcal{P}_{\mathrm{env}}(t)\\ &+ \Big(\frac{k_T}{C_{\mathrm{env}}} - \frac{1}{R_{\mathrm{env}}C_{\mathrm{env}}}\Big)\mathcal{T}_{\mathrm{env}}(t) + \frac{k_C}{C_{\mathrm{env}}}. \quad (5)\end{aligned}$$

If we know the temperature of the environment and CPU at some initial time $t_0 \le t$, then we can derive following Equations from (3) and (5):

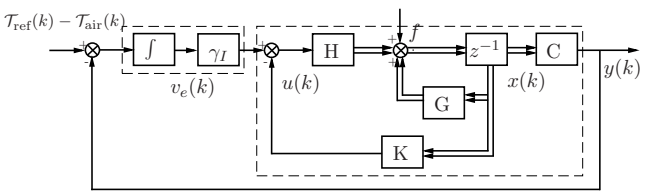


Fig. 3: The thermal control design with state feedback and integral actuator

$$\mathcal{T}_{\mathrm{cpu}}(t) = \int_{t_0}^{t}\sigma_1\mathcal{P}_{\mathrm{cpu}}(s)e^{-(t-s)\beta_1}ds + \mathcal{T}_{\mathrm{cpu}}(t_0)e^{-(t-t_0)\beta_1}, \quad (6)$$

$$\begin{aligned}\mathcal{T}_{\mathrm{env}}(t) &= \int_{t_0}^{t}\sigma_2\Big(\mathcal{P}_{\mathrm{env}}(s) + \mathcal{P}_{\mathrm{cpu}}(s)\Big)e^{-(t-s)\beta_2}ds\\ &+ \mathcal{T}_{\mathrm{env}}(t_0)e^{-(t-t_0)\beta_2}. \quad (7)\end{aligned}$$

where $\beta_1 \stackrel{\mathrm{def}}{=} (\frac{1}{R^{\mathrm{d}}_{\mathrm{cpu}}} + \frac{1}{R^{\mathrm{l}}_{\mathrm{cpu}}} - k_T)\cdot\frac{1}{(C^{\mathrm{d}}_{\mathrm{cpu}}+C^{\mathrm{l}}_{\mathrm{cpu}})}$, $\beta_2 \stackrel{\mathrm{def}}{=} \frac{1}{R_{\mathrm{env}}C_{\mathrm{env}}} - \frac{k_T}{C_{\mathrm{env}}}$, and $\sigma_2 \stackrel{\mathrm{def}}{=} \frac{1}{C_{\mathrm{env}}}$. According to the Figure 2 shown above, the absolute CPU temperature can be calculated as $\mathcal{T}_{\mathrm{cpu}}(t) + \mathcal{T}_{\mathrm{env}}(t) + \mathcal{T}_{\mathrm{air}}(t)$.

## V. Controller Design

We first present the standard state-space model used in control theory in Section V-A. In Section V-B, we design a thermal controller assuming that an ideal system with continuous power modes. In Section V-C, we will extend the controller design to a processor with only active/inactive power modes.

### A. State-Space Model Basics

We use the standard state-space model to represent continuous-time (ideal) system

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t) + f,\\ y(t) &= Cx(t), \quad (8)\end{aligned}$$

where $x(t), u(t)$, and $y(t)$ represent the state vector, the input vector, and the output vector, respectively. $A, B$, and $C$ represent the system matrices and $f$ represents a constant vector. Both the state matrices and constant vector are time-invariant quantities.

Since we have a computer-controlled discrete-time system, we will use following state-space mode for the discrete-time controller for active/inactive modes. For a sampling interval $T_s$, $u(t)$ is a constant and the sampled system of Equation (8) is

$$\begin{aligned}x((k+1)T_s) &= Gx(kT_s) + Hu(kT_s) + \tilde{f},\\ y(kT_s) &= \tilde{C}x(kT_s), \quad (9)\end{aligned}$$

where $G = e^{AT_s}$, $H = \int_0^{T_s} e^{At}Bdt$, $\tilde{C} = C$, and $\tilde{f} = \int_0^{T_s} e^{At}fdt$. The term $e^{At}$ can be computed by $\mathcal{L}^{-1}\{(sI -$

$A)^{-1}\}$, where $\mathcal{L}^{-1}$ is the inverse Laplace transform. In the remainder of the document, we abuse the notation by representing $x(kT_s)$ as $x(k)$, $x((k+1)T_s)$ as $x(k+1)$, $u(kT_s)$ as $u(k)$, and $y(kT_s)$ as $y(k)$. The above definitions may be found in any textbook on discrete-time control theory [29].

### B. Continuous Power Modes

As a first step towards our goal of designing a control-theoretic framework for thermal stress analysis, we employ *linear quadratic (LQ) optimal control* for real-time thermal management. Our design consists of an optimal state feedback and a servo that regulates the dynamics of the system. An LQ controller enables us to design an efficient and low-overhead controller, derive the feedback parameters before runtime (used in thermal-resiliency analysis), and smoothly track our reference input. In the future, we plan on applying more complex and robust controllers (e.g., $\mathcal{H}_\infty$ controllers) to decrease the controller's sensitivity to modeling inaccuracy and noise. However, as observed in the simulations and experiments of Section VII, our current LQ design is appropriately responsive to changes in environmental temperature.

In our system model, we specify the thermal power of the CPU as the control to the system. The controller is required to work as a servo and should follow the temperature reference, $\mathcal{T}_{\text{ref}}$. In our design, we consider $\mathcal{T}_{\text{cpu}}(t)$ as one of the variable to be controlled and $\mathcal{P}^{\text{d}}_{\text{cpu}}(t)$ as a manipulated variable (equivalent to $y(t)$ and $u(t)$, respectively, in continuous state-space model). The basic control structure is given in Figure 3.

From Equations (3) and (5), the continuous-time state space model can be written as

$$\begin{bmatrix}\dot{\mathcal{T}}_{\text{cpu}}(t)\\ \dot{\mathcal{T}}_{\text{env}}(t)\end{bmatrix} = \begin{bmatrix}-\beta_1 & k_T\sigma_1\\ k_T\sigma_2 & -\beta_2\end{bmatrix}\begin{bmatrix}\mathcal{T}_{\text{cpu}}(t)\\ \mathcal{T}_{\text{env}}(t)\end{bmatrix} + \begin{bmatrix}\sigma_1\\ \sigma_2\end{bmatrix}\mathcal{P}^{\text{d}}_{\text{cpu}}(t) + \begin{bmatrix}0\\ \sigma_2\end{bmatrix}\mathcal{P}_{\text{env}}(t). \quad (10)$$

While our analysis below is in the continuous-time domain, a discrete-time control system approach would be applied in an actual computer implementation. Therefore, we now note that we may easily convert the continuous-state space model to the discrete-time sampled system, $x(k+1) = Gx(k)+Hu(k)+f$ from the continuous-time state matrices $A = \begin{bmatrix}-\beta_1 & k_T\sigma_1\\ k_T\sigma_2 & -\beta_2\end{bmatrix}$ and $B = \begin{bmatrix}\sigma_1\\ \sigma_2\end{bmatrix}$ where $k$ is the sampling index, $T_s$ is sampling interval, and $G$ and $H$ can be calculated as described in Section V-A. For our given system, $x(k) \equiv \begin{bmatrix}\mathcal{T}_{\text{cpu}}(k)\\ \mathcal{T}_{\text{env}}(k)\end{bmatrix}$ and $u(k) \equiv \begin{bmatrix}\mathcal{P}_{\text{cpu}}(k)\end{bmatrix}$ where we are again abusing notation for the $\mathcal{T}$ and $\mathcal{P}$ functions.

To eliminate steady state tracking error, we design our system as a servo with an integrator. Define an additional error vector $v_e(t)$ in continuous time as,

$$\begin{aligned} v_e(t) &\stackrel{\text{def}}{=} \int_0^t (\mathcal{T}_{\text{ref}} - \mathcal{T}(t) - \mathcal{T}_{\text{air}}(t))dt \\ \dot{v}_e(t) &\stackrel{\text{def}}{=} \mathcal{T}_{\text{ref}} - \mathcal{T}_{\text{air}}(t) - \mathcal{T}(t) \\ &= -C\begin{bmatrix}\mathcal{T}_{\text{cpu}}(t)\\ \mathcal{T}_{\text{env}}(t)\end{bmatrix} + \mathcal{T}_{\text{ref}} - \mathcal{T}_{\text{air}}(t) \end{aligned} \quad (11)$$

where $C = [1, 1]$. Then, the system input is calculated with a gain $K_o = [\gamma_1, \gamma_2]$ and integral constant $\gamma_I$ in the following equation.

$$\begin{aligned} \mathcal{P}^{\text{d}}_{\text{cpu}}(t) &= -K_o\begin{bmatrix}\mathcal{T}_{\text{cpu}}(t)\\ \mathcal{T}_{\text{env}}(t)\end{bmatrix} + \gamma_I v_e(t) \\ &= -\Big((\gamma_1)\mathcal{T}_{\text{cpu}}(t) + (\gamma_2)\mathcal{T}_{\text{env}}(t)\Big) \\ &\quad +\gamma_I\int_0^t (\mathcal{T}_{\text{ref}} - \mathcal{T}_{\text{air}}(t) - \mathcal{T}_{\text{cpu}}(t) - \mathcal{T}_{\text{env}}(t))dt. \end{aligned} \quad (12)$$

We employ standard techniques from optimal control theory to derive $K_o$ and $\gamma_I$ and prove stability. Details are presented in an appendix of an extended version of this paper [11].

### C. Active/Inactive Power Modes

Since the CPU power cannot be varied continuously, the controller designed in the previous section cannot be directly applied to the setting of discrete active/inactive power modes. In this section, we extend the design of the continuous power modes controller described in the previous section to the active/inactive power mode setting by applying pulse-width modulation (PWM) techniques. Recall in Section IV that we stated the active/inactive power modes will be modeled via the thermal-aware periodic resource model with parameters $\Pi$ and $\Theta$. Thus, to control the system via this model, we must choose the appropriate values of $\Pi$ and $\Theta$. The $\Pi$ value is a design parameter which may be chosen at controller design-time and will be assumed fixed throughout controller execution. Typically, a smaller value of $\Pi$ will increase the system schedulability; however, a larger value of $\Pi$ will decrease the overhead potentially incurred by switching between the active and inactive power modes. (See Ahmed et al. [23] for algorithms for determining $\Pi$ in the thermal setting). The only constraint that our framework places on the chosen value of $\Pi$ is that it must evenly divide the sampling interval length $T_s$ (i.e., $T_s = \kappa\Pi$ for some $\kappa \in \mathbb{N}^+$).

Since we have only two power modes, we cannot arbitrarily set the power level. However, we may change the assigned resource capacity between sampling periods to approximate arbitrary power levels. Therefore, the assigned resource capacity will be the manipulated variable in our PWM system. Let $\Theta(k)$ denote the value of the resource capacity over the $k$'th sampling period. For determining the $\Theta(k)$ value, we use a method based on the *principle of equivalent areas* (PEA) for converting any arbitrary input signal into an equivalent PWM signal [30]. First, note that in a discrete-time system using *zero-order hold* (ZOH), the input signal is held constant over the sampling period. Specifically, for the $k$'th sampling

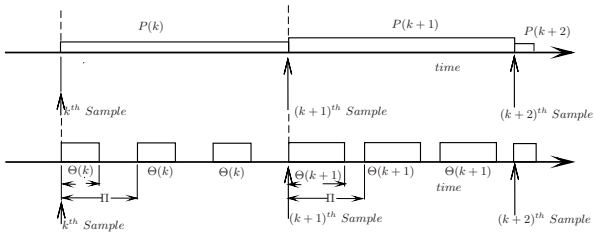


Fig. 4: The simplified power and modulation relationship

interval, the input $\mathcal{P}^{\text{d}}_{\text{cpu}}(k)$ is held over the $T_s$-length interval, resulting in a total energy dissipation of $T_s \cdot \mathcal{P}^{\text{d}}_{\text{cpu}}(k)$ over the interval. To get the equivalent area (i.e., energy) as the (ideal) system with continuous power modes, we must set $\Theta(k)$ such that the periodic modulations between the power modes of $\mathcal{P}_{\text{act}}$ and $\mathcal{P}_{\text{inc}}$ dissipate the equivalent amount of energy over the $T_s$-length interval. Figure 4 illustrates the area equivalence between the continuous and PWM controllers. The appendix of the extended version of this paper [11] describes how to choose $T_s$ to minimize error due to the PWM approximation.

More formally, we may derive the following relationship between $\mathcal{P}^{\text{d}}_{\text{cpu}}(k)$ and $\Theta(k)$,

$$\begin{aligned} \kappa\Pi\mathcal{P}^{\text{d}}_{\text{cpu}}(k) &= \kappa\Big(e_{\text{act}} + \int_0^{\Theta(k)} \mathcal{P}_{\text{act}}dt + e_{\text{inc}} + \int_{\Theta(k)}^{\Pi} \mathcal{P}_{\text{inc}}dt\Big) \\ \Rightarrow \mathcal{P}^{\text{d}}_{\text{cpu}}(k) &= \left(\frac{\mathcal{P}_{\text{act}} - \mathcal{P}_{\text{inc}}}{\Pi}\right)\Theta(k) + \mathcal{P}_{\text{inc}} + \frac{1}{\Pi}(e_{\text{act}} + e_{\text{inc}}). \end{aligned} \tag{13}$$

**Algorithm 1** Control Algorithm

**Require:** Reference Temperature $\mathcal{T}_{\text{ref}}$; Feedback Gain $K \equiv [\gamma_1, \gamma_2]$; Integral Constant $\gamma_I$; PWM Period $\Pi$; Number of PWM periods in a sampling period $\kappa$.

1: **while** At beginning of sampling period $[t_\ell, t_{\ell+1}) : t_\ell \equiv \kappa\ell\Pi$ **do**
2: Sample $\mathcal{T}_{\text{cpu}}(t_\ell) + \mathcal{T}_{\text{env}}(t_\ell) + \mathcal{T}_{\text{air}}(t_\ell)$.
3: $\dot{v}_e(t_\ell) = \mathcal{T}_{\text{ref}} - (\mathcal{T}_{\text{cpu}}(t_\ell) + \mathcal{T}_{\text{env}}(t_\ell) + \mathcal{T}_{\text{air}}(t_\ell))$
4: $Tot\_\dot{v}_e(t_\ell) = Tot\_\dot{v}_e(t_{\ell-1}) + \gamma_I\kappa\Pi\frac{\left(\dot{v}_e(t_\ell)+\dot{v}_e(t_{\ell-1})\right)}{2}$
5: $\mathcal{P}_{\text{cpu}}(t_\ell) = \Big(Tot\_\dot{v}_e(t_\ell) - \big(\gamma_1\mathcal{T}_{\text{cpu}}(t_\ell) + \gamma_2\mathcal{T}_{\text{env}}(t_\ell)\big)\Big)$
6: $\Theta(t_\ell) = \min\left(\Pi \times \frac{(\mathcal{P}_{\text{cpu}}(t_\ell)-\mathcal{P}_{\text{inc}})}{\mathcal{P}_{\text{act}}-\mathcal{P}_{\text{inc}}}, \Pi\right)$
7: $i = \max\{j \in \mathbb{Z}_{q+1} \mid \Theta^{(j)} \leq \Theta(t_\ell)\}$
8: Update real-time performance mode to $M_i$.
9: Set PWM to operate at period of $\Pi$ and width of $\Theta(t_\ell)$.
10: **end while**

The PWM controller pseudocode is presented in Algorithm 1. The controller proposed here consists of two integrated operations: the thermal controller and the PWM modulator. The first step is to obtain the sample CPU temperature (Line 2 of Algorithm 1). The error is then calculated by taking the difference between the reference temperature and the CPU temperature (Line 3). The error is integrated into the error vector and added to vector sum of the integrated error in the next line (Line 4). After which, the power input is calculated (Line 5) and the equivalent $\Theta$ is calculated from the property of Equation (13) (Line 6). Finally, the appropriate mode is selected (Line 7), the mode change is performed (Line 8), and the pulse-width modulator is invoked for the next $\kappa$ $\Pi$-length intervals (Line 9). It is important to note that $\Theta(t_\ell)$ calculated in Line 6 does not have to be equal the $\Theta^{(j)}$ for the selected mode; we must only select the highest mode with $\Theta^{(j)} \leq \Theta(t_\ell)$. (If $\Theta(t_\ell)$ is larger, we are only giving the mode more processing than it requires.) It should also be observed that all operations, except for finding the appropriate mode, may be done in $O(1)$ time. Finding the highest real-time performance mode that may execute can be done in $O(\lg q)$ time (via binary search) where $q$ is the number of real-time performance modes.

## VI. Thermal-Resiliency Calculation

In this section, we explain how to derive the real-time thermal resiliency $\Lambda(M_i, \mathcal{T}_{\text{ref}})$ for a given real-time performance mode $M_i$ and reference temperature $\mathcal{T}_{\text{ref}}$. Assuming a steady-state error of zero, we will now briefly outline how to obtain a solution for $\Lambda(M_i, \mathcal{T}_{\text{ref}})$.[3] Assume that we have reached the steady-state by the $(k-1)$'th sampling period. Therefore, $\mathcal{T}_{\text{cpu}}(k) = \mathcal{T}_{\text{cpu}}(k-1)$, $\mathcal{T}_{\text{env}}(k) = \mathcal{T}_{\text{env}}(k-1)$, $\mathcal{T}_{\text{air}}(k) = \mathcal{T}_{\text{air}}(k-1)$, and $\Theta(k) = \Theta(k-1)$. Substituting the temperature equalities into Equations (6) and (7) allows us to solve for $\mathcal{T}_{\text{cpu}}(k)$ and $\mathcal{T}_{\text{env}}(k)$ to obtain a function of $\mathcal{T}_{\text{air}}(k)$, $\mathcal{T}_{\text{ref}}$, and $\Theta(k)$. Since we are interested in obtaining $\Lambda(M_i, \mathcal{T}_{\text{ref}})$, we may fix $\mathcal{T}_{\text{ref}}$ and $\Theta(k) = \Theta^{(i)}$ Since the steady-state error is zero, we also have

$$\mathcal{T}_{\text{ref}} = \mathcal{T}_{\text{cpu}}(k) + \mathcal{T}_{\text{env}}(k) + \mathcal{T}_{\text{air}}(k). \tag{14}$$

Combining Equation 14 with the function of $\mathcal{T}_{\text{air}}(k)$ obtained from $\mathcal{T}_{\text{cpu}}(k)$ and $\mathcal{T}_{\text{env}}(k)$ allows us to solve for $\mathcal{T}_{\text{air}}(k)$. Thus, solving the entire system results in a value for $\mathcal{T}_{\text{env}}(k)+\mathcal{T}_{\text{air}}(k)$ (i.e., value of $\Lambda(M_i, \mathcal{T}_{\text{ref}})$). The resulting expression is quite complicated as it requires solutions to second-order inhomogeneous equations. Full details and the closed-form expression for $\Lambda(M_i, \mathcal{T}_{\text{ref}})$ are provided in the extended paper [11].

## VII. Validation

In this section, we evaluate our control framework both in simulations and upon an experimental hardware testbed.

### A. Simulations

In the simulations, we simulate the execution of a single-core processor which consists of a thermal controller, PWM frequency controller loop, and scheduling algorithm. The following task parameters are used in our simulations:

- Each sporadic task $\tau_j = (e_j, d_j, p_j)$ has a period $p_j$ uniformly drawn from the interval $[5, 15]$. (A small period range is used to keep $LCM$ of periods from becoming too

[3] The approach may be generalized when there is bounded steady-state error. However, the approach will be similar, and we omit the details due to space.

TABLE I: Testbed Parameters

| Parameter | Variable | Value |
|---|---|---|
| CPU Active Power | $\mathcal{P}_{\text{act}}$ | 73 $W$ |
| CPU Idle Power | $\mathcal{P}_{\text{inc}}$ | 20 $W$ |
| Server Period | $\Pi$ | 20 ms |
| Sampling Time | $T_s$ | 100 ms |
| Optimal Feedback | $K_o$ | $\begin{bmatrix}.5725 & 0\end{bmatrix}$ |
| Q matrix in Performance Index | Q | $\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$ |
| R matrix in Performance Index | R | $\begin{bmatrix}1\end{bmatrix}$ |
| Integral Gain | $\gamma_I$ | 0.00042 |

large). The execution time requirement $e_j$ set to the task utilization times $p_j$, where task utilization is calculated using the UUnifast algorithm [31]. For each task, $d_j$ equals $p_j$. The tasks are scheduled by EDF.

- The total number of tasks is eight; each task $\tau_j$ has three different real-time performance modes where $\tau_j^{(2)} = (e_j, d_j, p_j)$; $\tau_j^{(1)} = (.2e_j, d_j, p_j)$; and $\tau_j^{(0)}$ means that task is not selected. From set of all possible combinations of tasks, we have selected fifteen modes with utilizations ranging from zero to one.

We refer to the controller described in Algorithm 1 as *Temperature Regulated Capacity Bound* (TRCB). In our simulations, we closely compare the performance of our proposed method with [8] referred to as *Thermal Control Utilization Bound* (TCUB). TCUB has been chosen due to its low controller time complexity of $O(1)$. TCUB works by attempting to track a reference temperature and adjusting system utilization as needed by changing task modes via a mode assignment heuristic. The major difference between TCUB and TRCB is that TCUB does not have predefined modes. Therefore, TCUB may differ in the assigned modes from run to run for the same system temperature. Furthermore, TCUB does not use multiple power levels. TRCB on the other hand has predefined modes which permit the derivation of thermal resiliency for each mode. TCRB also utilizes a low-power mode (if available).

In our simulation, we use the same system parameters as our testbed (Intel Pentium IV 3.0 GHz). The pertinent power and control parameters are given in Table I. Extensive testbed runs were carried out to generate the remaining system parameters using SI. We use the SI tools provided by Matlab to derive the system state-space parameters. Also we use the system parameters, generated from our testbed as the simulation parameters. We observe a matching of our testbed readings and the simulation. More details on this process are contained in the extended version of the paper [11].

In Figure 5, the system response and the utilization has been shown for both TRCB (right graphs) and TCUB (left graphs) given a stable air temperature $\mathcal{T}_{\text{air}}$ temperature equal to $5°C$. The behavior of both controllers in this stable environment is nearly identical for thermal and utilization behavior. (The difference is due to the fact that TRCB uses EDF and TCUB uses RM scheduling). For TRCB, we also display the achieved modes at any given time in the simulation in the lower right

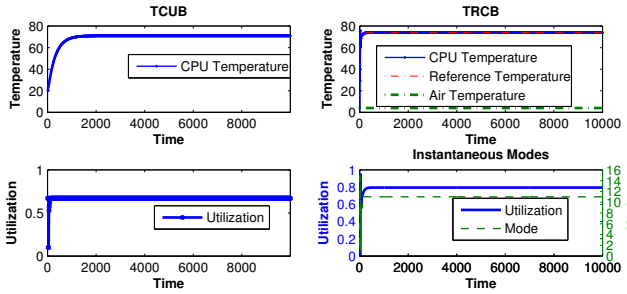


Fig. 5: Fixed $\mathcal{T}_{\text{air}}$ for Simulation. Left plots represent TCUB and right plots represent TCRB.

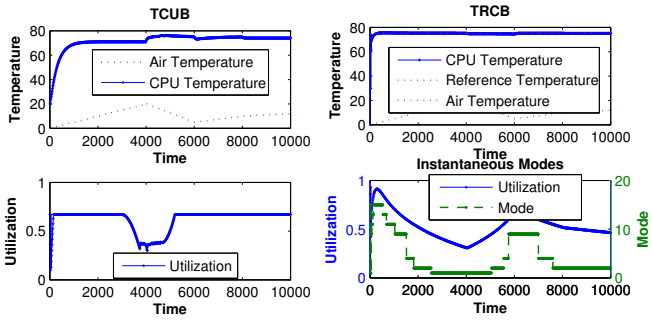


Fig. 6: Dynamically Varying $T_{air}$ for Simulation. Left plots represent TCUB and right plots represent TCRB.

graph.

Figure 6 shows the behavior of both TRCB and TCUB when $\mathcal{T}_{\text{air}}$ is dynamically changed over time. In the top two graphs of the figure, the absolute CPU temperatures over time obtained by TCUB and TRCB, respectively, are plotted along with the $\mathcal{T}_{\text{air}}$. The two bottom graphs of Figure 6 present the achieved utilization for each controller; additionally, the bottom right graph displays the active mode at any point in time for TRCB. Observe that both controllers are able to track the reference temperature $\mathcal{T}_{\text{ref}}$ despite the sharp changes in $\mathcal{T}_{\text{air}}$. For both controllers, the utilization appropriately tracks the changes in air temperature. When the air temperature increases, both controllers decrease the system utilization and increase the utilization again when the air temperature drops. Similarly, the mode plot in the lower right graph tracks the temperature changes.

Regarding the real-time performance, figures displaying deadline miss ratios have been omitted as no deadline miss was experienced for either controller in all the simulations. TCUB uses a safe utilization bound of approximately $67\%$ to make deadline misses improbably for rate-monotonic scheduling [26]. However, TCRB guarantees that no deadlines are ever missed due to verification using a multi-modal schedulability test [27] as described in Section IV-B.

Thus far, the empirical performance of TRCB and TCUB may appear similar. However, we believe the distinguishing feature of TRCB is the ability to guarantee hard deadlines and to calculate thermal resiliency levels during design time. Ther-

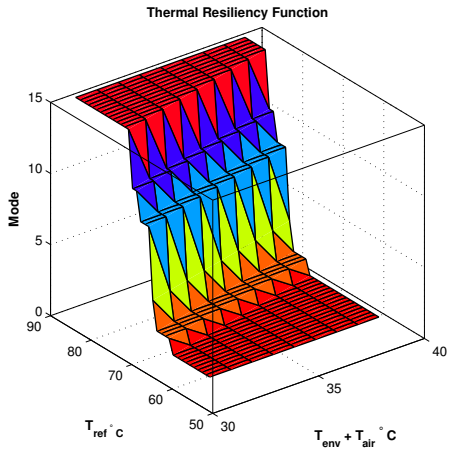


Fig. 7: Thermal resiliency over modes and $\mathcal{T}_{\text{ref}}$.

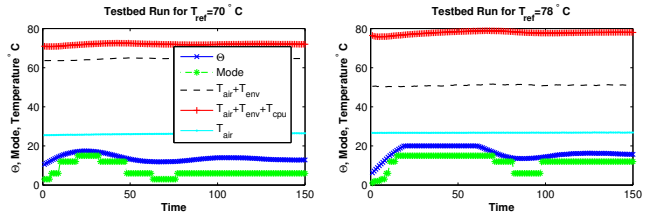


Fig. 8: The testbed running at different $\mathcal{T}_{\text{ref}}$ values showing the $\Theta$ and Mode change over the time

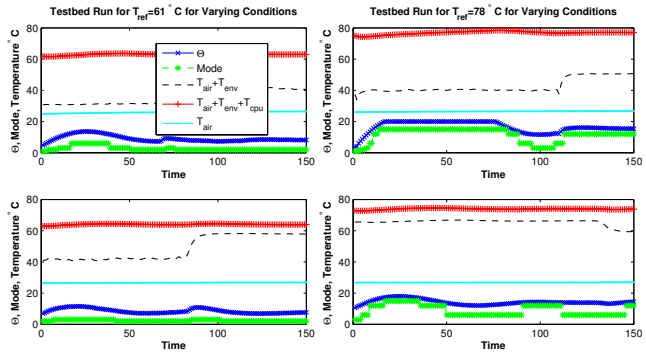


Fig. 9: The testbed running at varying environmental conditions showing the $\Theta$ and Mode change over the time

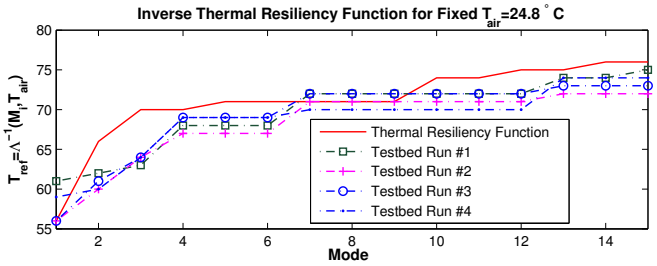


Fig. 10: Thermal Resiliency for the Simulation.

mal resiliency calculation provides a *non-destructive* thermal stress analysis for real-time performance modes in an unpredictable operating environment. Our approach has achieved the ability to calculate the thermal resiliency by forcing the system to execute in a very predictable manner (i.e., periodic executions from PWM). To evaluate and illustrate our thermal resiliency calculation, we have used the technique in Section VI to calculate the thermal resiliency levels for our randomly-generated multi-mode system. Figure 7 displays the thermal resilience $\Lambda(M_i, \mathcal{T}_{\text{ref}})$ for a range of modes and reference temperatures. Observe that the thermal resiliency increases with decreasing modes or increasing $\mathcal{T}_{\text{ref}}$.

### *B. Experiments upon Hardware Testbed*

To further confirm the validity of the theoretical results, we have run a task system with eight tasks, each with three modes (identical to the simulation setting), on our hardware testbed. Each task performs numerical calculations while executing on the system. Our hardware testbed behaves similar to the simulations of the previous subsection. Figure 8 presents testbed runs for a fixed air and environment temperature. Figure 9 shows how the testbed behaves when an outside heat source is dynamically introduced into the environment. Observe that there is a momentary drop in performance mode; however, the system soon stabilizes.

Finally, we validate our thermal resiliency calculation. Unfortunately, we do not have test equipment to accurately vary the air or environment temperature. Thus, we consider the air temperature to be fixed at the room temperature (in this case $\mathcal{T}_{\text{air}} = 24.8°C$). Instead, we indirectly analyze the thermal resiliency function via the inverse of the thermal resiliency function $\Lambda^{-1}(M_i, \mathcal{T}_{\text{air}}) = \min\{\mathcal{T}_{\text{ref}} \mid \mathcal{T}_{\text{air}} \leq \Lambda(M_i, \mathcal{T}_{\text{ref}})\}$. Intuitively, a lower value of $\Lambda^{-1}(M_i, \mathcal{T}_{\text{air}})$ means the system can operate at a lower temperature and thus is more resilient than a higher value of the function. We have calculated this function for four different runs of the hardware testbed (to ensure that minor fluctuations of the air temperature do not affect the system). Figure 10 shows a plot of the thermal resiliency of the testbed runs when the $\mathcal{T}_{\text{ref}}$ is changed. The figure shows that the calculated inverse resiliency of the system increases with increasing operating mode. Most importantly, the calculated thermal resiliency tracks the actual behavior of the testbed and provides a safe upper bound on $\mathcal{T}_{\text{ref}}$ in a large majority of the cases which validates the effectiveness of the resiliency function.

## VIII. Conclusions

In this paper, we have addressed the problem of obtaining performance guarantees in an unpredictable thermal environment. Towards this challenge we have presented a control-theoretic framework for thermal stress analysis in real-time systems. Our proposed method employs a nested feedback control system, which is based on optimum control theory. For our system, we derive strong thermal-resiliency and hard-real-time guarantees for any real-time performance mode. Our method has the distinct advantage of being able to verify the real-time thermal resiliency of a system before it is put into operation. In addition, we show via simulations that our framework performs as well as previous approaches which have no formal guarantee on the thermal resiliency. Our im-

plementation upon a hardware testbed validates our proposed model and control framework.

In future work, we plan to extend our framework to control designs that are more robust to model inaccuracies (e.g., $\mathcal{H}_\infty$ or model-predictive controllers). As a initial step in designing a framework for thermal stress analysis, our current design uses two RC circuits (for dynamic and leakage currents) to model the CPU temperature. We plan on extending our model to permit multiple RC circuits for heterogeneous thermal distributions and generalizing our thermal equations for more complex RC circuit layouts. We hope to derive a general-theoretic design framework that captures "resiliency" metrics for other system properties (e.g., energy, noise, etc.) and extend our analysis to other hardware settings (e.g., multicore, DVS).

## ACKNOWLEDGMENTS

This research has been supported in part by the NSF (Grant Nos. CNS-0953585, CNS-1116787, and CNS-1136007), the Air Force Office of Scientific Research (Grant No. FA9550-10-1-0210), and two grants from Wayne State University's Office of Vice President of Research.

## REFERENCES

[1] J. Sergent and A. Krum, *Thermal Management Handbook for Electronic Assemblies*. McGraw-Hill Professional, 1998.

[2] S. Kim, P. Tathireddy, R. Normann, and F. Solzbacher, "Thermal impact of an active 3-d microelectrode array implanted in the brain," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 15, no. 4, pp. 493–501, December 2007.

[3] G. Lazzi, "Thermal effects of bioimplants," *IEEE Engineering in Medicine and Biology Magazine*, vol. 24, no. 5, pp. 75–81, September - October 2005.

[4] J. C. LaManna, K. A. McCracken, M. Patil, and O. J. Prohaska, "Stimulus-activated changes in brain tissue temperature in the anesthetized rat," *Metabolic Brain Disease*, vol. 4, no. 4, pp. 225–237, 1989.

[5] P. Ruggera, D. Witters, G. von Maltzahn, and H. Bassen, "*In vitro* assessment of tissue heating near metallic medical implants by exposure to pulsed radio frequency diathermy," *Physics in Medicine and Biology*, vol. 48, no. 17, pp. 2919–2928, 2003.

[6] G. Kelly, "Body temperature variability (part 1): a review of the history of body temperature and its variability due to site selection, biological rhythms, fitness, and aging," *Alternative Medicine Review*, vol. 11, no. 4, pp. 278–293, 2006.

[7] N. Timmons and W. Scanlon, "An adaptive energy efficient mac protocol for the medical body area network," in *1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology, 2009*, May 2009, pp. 587 – 593.

[8] Y. Fu, N. Kottenstette, Y. Chen, C. Lu, X. D. Koutsoukos, and H. Wang, "Feedback thermal control for real-time system," in *Proceedings of the Real-Time and Embedded Technology and Applications Systems Symposium*. Stockholm, Sweden: IEEE Computer Society Press, April 2010.

[9] X. Fu, X. Wang, and E. Puster, "Simultaneous thermal and timeliness guarantees in distributed real-time embedded systems," *Journal of Systems Architecture*, 2010, to Appear.

[10] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A resource allocation model for qos management," in *Proceedings of the 18th IEEE Real-Time Systems Symposium*, ser. RTSS '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 298–. [Online]. Available: http://portal.acm.org/citation.cfm?id=827269.828990

[11] P. M. Hettiarachchi, N. Fisher, M. Ahmed, L. Y. Wang, S. Wang, and W. Shi, "The design and analysis of thermally-resilient hard-real-time systems (extended version)," Wayne State University, Tech. Rep., 2011, available at http://www.cs.wayne.edu/~fishern/papers/thermal-control-rtas2012.pdf.

[12] A. K. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983, available as Technical Report No. MIT/LCS/TR-297.

[13] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *International Symposium on High-Performance Computer Architecture*, 2001.

[14] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *International Symposium on Computer Architecture*, 2003.

[15] N. Bansal and K. Pruhs, "Speed scaling to manage temperature," in *Symposium on Theoretical Aspects of Computer Science*, 2005.

[16] S. Wang and R. Bettati, "Reactive speed control in temperature-constrained real-time systems," *Real-Time Systems Journal*, vol. 39, no. 1-3, pp. 658–671, 2008.

[17] J.-J. Chen, S. Wang, and L. Thiele, "Proactive speed scheduling for frame-based real-time tasks under thermal constraints," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2009.

[18] G. Quan and Y. Zhang, "Leakage Aware Feasibility Analysis for Temperature-Constrained Hard Real-Time Periodic Tasks," in *Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems-Volume 00*. IEEE Computer Society, 2009, pp. 207–216.

[19] J.-J. Chen, C.-M. Hung, and T.-W. Kuo, "On the minimization of the instantaneous temperature for periodic real-time tasks," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2007.

[20] T. Chantem, R. P. Dick, and X. S. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," in *Design, Automation and Test in Europe*, 2008.

[21] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele, "Thermal-aware global real-time scheduling on multicore systems," in *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE Computer Society Press, April 2009.

[22] A. Ferreira, D. Mosse, and J. Oh, "Thermal faults modeling using a rc model with an application to web farms," in *Proceedings of the Euromicro Conference on Real-Time Systems*. IEEE Computer Society, July 2007.

[23] M. Ahmed, N. Fisher, S. Wang, and P. Hettiarachchi, "Minimizing peak temperature in embedded real-time systems via thermal-aware periodic resources," *Sustainable Computing: Informatics and Systems*, vol. 1, no. 3, pp. 226 – 240, 2011.

[24] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, April 2008.

[25] *Intel Pentium 4 processor in the 423-pin package thermal design guidelines,*. Intel Corp., 2000.

[26] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[27] N. Fisher and M. Ahmed, "Tractable real-time schedulability analysis for mode changes under temporal isolation," in *Proceedings of the 9th IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTImedia)*. IEEE Computer Society, October 2011.

[28] Y. Liu, R. P. Dick, L. Shang, and H. Yang, "Accurate temperature-dependent integrated circuit leakage power estimation is easy," in *Proceedings of the conference on Design, automation and test in Europe*, Nice, France, 2007, pp. 1526–1531.

[29] K. Ogata, *Discrete-time control systems (2nd ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995.

[30] A. K. Gelig and . Churilov, Alexander N., *Stability and oscillations of nonlinear pulse-modulated systems / Arkadii Kh. Gelig, Alexander N. Churilov*. Boston : Birkhauser, 1998, includes bibliographical references (p. [343]-359) and index.

[31] E. Bini and G. Buttazzo, "Biasing effects in schedulability measures," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems*. IEEE Computer Society, 2004, pp. 196–203.