Event-Based Power/Performance-Aware Thermal Management for High-Density Microprocessors

Alberto Leva, Federico Terraneo, Irene Giacomello, and William Fornaciari

***Abstract*— The density of modern microprocessors is so high, that operating all their units at full power would destroy them by thermal runaway. Hence, thermal control is vital, but at the same time has to integrate with power/performance management, to not unduly limit computational speed. In addition, the controller must be simple and computationally light, as millisecond-scale response is required. Finally, since microprocessors face a variety of operating conditions, postsilicon tuning is an issue. We here present a solution, by exploiting event-based control and a hardware/software partition to maximize efficiency, lightness, and flexibility. We show experiments on real hardware, evidencing the obtained advantages over the state of the art.**

***Index Terms*— Dark silicon, event-based control, microprocessors, power/performance management, temperature control.**

I. Introduction

IN THE last years, the power density of microprocessors (hereinafter μPs for short) has been increasing dramatically [1]. This has led to the so-called "dark silicon" problem [2], which is the impossibility of operating all the units of a μP at full power without destroying the device by thermal runaway. In addition, modern μPs contain multiple cores and run a huge number of tasks with a very time-varying resource usage. Hence, not only the maximum consumable power can burn the chip, but the chip's power consumption is subject to large, abrupt, and hardly predictable variations—a *scenario* that made some researchers doubt on the future of multicore scaling [3]. In addition, as power densities get higher, thermal dynamics get faster. Some years ago, it was possible to control a μP's temperature with a fan. Nowadays, such an actuator is too slow. The only way to reduce thermal dissipation is to transiently diminish the computational power, for example, reducing the clock frequency. However, the clock frequency is also subject to power/performance management, that selects low frequencies when the system load is low, to save power, and high frequencies when the load is high, for a fast response.

On the one hand, therefore, an effective temperature control is vital for a μP, but on the other hand, preserving thermal safety should not unduly affect computational speed. The thermal/performance tradeoff is nowadays a relevant research theme; a recent result on the matter is the "sprint computing" approach [4] that exploits thermal dynamics to transiently exceed a μP's thermal design power (TDP) without reaching unsafe temperatures. Sprint computing has found an application, as a hardware controller, in the Intel Turbo Boost 2.0 [5].

In this paper, building on the preliminary results of [6], we present a thermal/performance management solution that is easy to integrate with any power/performance policy. The presented solution exploits event-based control and a convenient hardware/software partition. Besides guaranteeing safe and effective operation, our solution has a very low computational overhead and a high configurability, to address the variety of deployment and operating conditions that a μP has to face.

This paper is organized as follows. Section II provides an overview of the problem, leading in Section III to motivate the proposed approach. Sections IV and V deal with the system model and the control synthesis, while Section VI describes the event-based realization. Section VII presents experimental results, comparing the proposed scheme with the state of the art. Section VIII draws some conclusions and outlines future research.

II. Problem Overview

In a μP, small volumes of active silicon release thermal power, depending on the computational load and on the clock frequency, to a chip-wide bulk substrate. There are also some direct connections among the active volumes, mainly via the conducting metal layers. The bulk exchanges through a metallic spreader with a sink that disperses heat into the environment. The objective of thermal control is to maintain the active silicon volumes at a safe temperature.

The temperatures of the active volumes swing against the local temperatures of the bulk, that in turn swing against that of the spreader and sink. The capacity of the active volumes is invariantly smaller than that of the bulk zones surrounding them, and the capacity of these is smaller than that of the spreader and sink. The overall thermal dynamics has thus three main time scales. From fastest to slowest, these correspond to active silicon, bulk, and spreader/sink.

Technology tends to scale down to smaller dimensions, which modifies the said time scales correspondingly. This modification can make previous control solutions unusable. In particular, to govern the fastest dynamics of active silicon, acting on the heat sink (e.g., with a fan) is nowadays too slow to be feasible. The only suitable actuators act on the clock;

Manuscript received September 21, 2016; revised January 31, 2017; accepted February 16, 2017. Date of publication March 30, 2017; date of current version February 8, 2018. Manuscript received in final form February 24, 2017. Recommended by Associate Editor Q. Wang.

A. Leva, F. Terraneo, and W. Fornaciari are with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, 20133 Milan, Italy (e-mail: alberto.leva@polimi.it; federico.terraneo@polimi.it; william.fornaciari@polimi.it).

I. Giacomello was with the Dipartimento di Elettronica, Informazione e Bioingegneria, 20133 Milan, Italy (e-mail: irene.giacomello@mail.polimi.it).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

dynamic voltage and frequency scaling (DVFS) modifies its frequency, whereas "clock gating" cuts off some of its cycles. Such actions, however, reduce both power dissipation and computational speed, making thermal management inherently coupled with that of power/performance.

The history of thermal/power/performance control in μPs can be broadly divided in three eras. The most ancient extends up to the introduction of the Intel Pentium; at the time power densities were so low, and geometries so large, that there was not even the need for a heat sink. Thermal control was no problem, and the only "power/performance" management was software tools to freeze the μP in the absence of load—a useful feature for laptop batteries those days. Research on the matter was practically absent.

Then, μPs started to need a heat sink, and the second era begun. However, the volumes and surfaces of active areas (thus, thermal capacities and conductances) were still large enough to allow operating all the units at full power. One had just to design the chip so as to sustain the worst possible thermal stress (which was feasible) and then provide decent dissipation conditions with the sink. From the application viewpoint, such a *scenario* separated thermal control from power/performance management. The former, given the long time scales, was dealt with by a fan. For the latter, software components were created (e.g., the Linux "governors") to act on DVFS based on the measured load. Research papers started appearing, but mostly from the computer engineering side. A pioneering work is [7], where possible actuators are discussed, and the idea appears of introducing control to handle a chip that cannot sustain all units at full power. However, for "control," the authors mean a threshold-based mechanism that intervenes only when a critical temperature is reached, and is generally stateless. Controllers in a more strict sense were introduced later on; for example, [8] proposes a PI-based solution (but does not discuss implementability issues).

We are now in the third era, the "dark silicon" one [3]. Thermal and power/performance management cannot be separated in frequency anymore. A certain divergence can be observed between academic research and the technologies introduced by the μP industry. Many research papers were published, with progressively more advanced control techniques, including linear-quadratic regulators [9], model predictive control [10]–[14], convex optimization [15], extremum seeking control [16], and much more. However, such advanced techniques are typically tested on purpose-specific simulators, and are very difficult to port to real hardware, due to limitations in terms of sensors and actuators. For example, at present, typical on-chip temperature sensors have a resolution of 1 °C and an even higher noise band, and many μPs have multiple cores, but one DVFS actuator for all of them. Moreover, advanced controls are often computationally heavyweight, which conflicts with the need to operate at the time scale now required, and many are centralized, adding the problem of moving data around the chip fast enough.

The industry has been facing the problem differently. Thermal control has been moved to hardware, using simple controllers with fixed and fast rates, activated only when a temperature threshold is trespassed; software power/performance management has been preserved. As for the actuators, DVFS is the preferred one for power/performance, whereas hardware thermal controllers can either take over DVFS, or act, e.g., by clock gating. This choice has solid motivations, the main one being to guarantee a safe thermal *protection*—rather than *control* in the full sense of the term—leaving total freedom to power/performance governors. However, for the same reasons, integration between the two functionalities is given up structurally, and thermal management has to incur the rigidity of hardware.

For the sake of completeness, we have to notice that in the literature alternatives to DVFS-based control can be found. The major one worth discussing is to control temperature via thread migration from "hot" to "cold" cores, taking care to maximize performance [17]–[19]. However, the time for a task migration can vary by orders of magnitude depending on the presence or absence of hardware support, and depends on unpredictable facts like the amount of memory/cache to move [20], [21]. Also, and most important, exploiting task migration means bringing into play relevant parts of the operating systems—most notably, the scheduler—that already have to obey a number of other constraints. Issues like this are not to be neglected in complex systems: the more functionalities a component is involved into, the more checks are in order when that component is maintained or redesigned. Since for example scheduling, power/performance and thermal control are typically managed by different developers, care has to be taken not to create undue couplings among the effects of their design choices. As such, we strongly prefer to aim at a solution that is as transparent and as agnostic as possible concerning the operating system, in particular not being affected by modifications in the scheduling and task migration policies.

III. Motivation of the Proposed Approach

Controlling the active silicon dynamics already requires millisecond-scale reaction. It is not totally clear how much innovative layouts like 3-D stacking [22] will exacerbate this, but the trend is evident [23], [24]. Complex fixed-rate controllers are thus problematic. If realized in software, they steal a relevant computational power. If made in hardware, they require silicon area, complicate the chip design, and affect consumption.

Moreover, when considering the integration with power/performance management, one has to remember that CPU load is necessarily measured over a time interval,[1] and this interval cannot be too small, or the measurement would be too noisy to be useful. The problem is that the time scale of the fast thermal dynamics is already comparable to sensible load measurement intervals, and will eventually become smaller. When this happens, the time scales of power/performance management, tied to the load measurement period, and of thermal control, tied to the active silicon thermal dynamics, will be flipped. To give just one example of the consequences, combining thermal/power/performance

[1] Avoiding complications useless herein, think of the load as the percent of the said interval in which the μP was not idle.

management by hooking thermal control to the load measurement period will become infeasible.

The mainstream industrial solution, as said earlier, is to leave power/performance to software and include hardware fixed-rate temperature controllers of simple structure, which act on the DVFS command at a millisecond scale, and are activated only when a *really* critical temperature is reached. For example, the hardware thermal protection of sixth generation core i5 Intel μPs is activated at 100 °C, which is well above "normal" operating temperatures.

In our opinion, this has two drawbacks: it prevents a coupled management of thermal and power/performance issues, and allows for limited tuning to face various installation conditions. It is worth noticing that in the past, for "installation conditions," one basically meant thermal exchanges: in this respect, relevant differences were the presence or absence of a fan, the casing and the like—in one word, what makes a server differ from a laptop, a tablet from a smartphone, and so forth. Nowadays, for tuning a thermal controller, the dynamics of the heat dissipation mechanism is so slow to be hardly relevant, no matter how that mechanism is realized. In addition, the small active silicon capacities do not filter out fast load variability as they did with larger geometries: if an application switches between high and low power request in milliseconds, with modern μPs, the resulting temperature swings are well visible. Therefore, to provide installation and operation adaptability, at present, one has to account for the type of applications, the operating system scheduling and resource allocation policy, load balancing, task migration, and similar components of a computing system—definitely, a variability that a software-configurable control is more keen to address successfully.

We believe that event-based control is the way to go. An event-based controller can react faster than a fixed-rate one, and do so only when needed. Its action can thus be effective at high frequency, where the capacity of the bulk practically decouples the dynamics of active silicon volumes—as will be shortly proved by experiments—and allows for a decentralized structure; at the same time, when no high-frequency activity is needed, such a controller requires a low computational effort. We further believe in the benefits of a hardware/software partition. The event triggering mechanism can be realized in hardware as a simple state machine, with minimum area and consumption, while the control law can be software, maximizing adaptability to the various installation and operating conditions.

IV. Controlled System Model

We start from some open-loop experimental results, obtained with an Intel core i5-6600K running Linux. The considered μP—the same used in the experimental setup of Section VII—has four cores, denoted in the following by Cores 0–3. We disabled all software thermal controls, leaving only the hardware protection, which, however, never intervened. We present four tests, summarized in the following.

1) We subjected Core 0 to an abrupt load step from 0% to 100%, at time 0.1 s, using the `cpuburn` thermal stresser [25], that fills the core pipelines with mathematical instructions. The other cores were left unloaded if not for the inevitable operating system activities. The frequency was kept fixed at its maximum value through the `userspace` governor [26].
2) We kept Core 0 at 100% load with `cpuburn` and Cores 1–3 unloaded, and applied a DVFS step (the 6600k has a single DVFS for all the cores) from the minimum to the maximum frequency, at time 0.1 s, with the `userspace` governor.
3) We repeated test 1 with a program we wrote and named `cachemiss`, that contrary to `cpuburn` occupies the core by just allocating and deallocating memory, with hardly any arithmetic/logic activity.
4) We repeated test 2 with `cachemiss`.

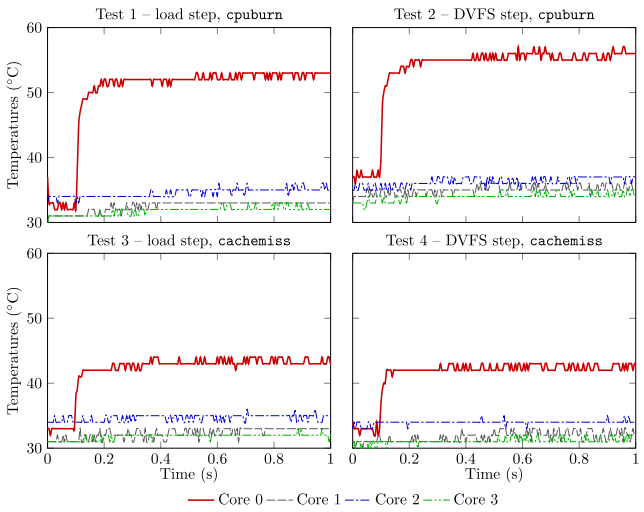

Fig. 1. Temperature responses in open-loop tests 1–4.

The responses of the core temperatures in the four tests, raw as coming from the on-chip sensors, are shown in Fig. 1.

We first observe that in all the tests, after the fast thermal transient, the temperature increment of Cores 1–3 is about 1/10 of that of Core 0. Also, though the measurement resolution is low, it seems that Core 0 responds as a first-order system, while Cores 1–3 exhibit a higher order dynamics (the figure shows just the initial part of the temperature response of Cores 1–3). Finally, a 0%–100% load step produces a quite different *power* step depending on the application, as shown by the different temperature variations produced on Core 0 by `cpuburn` and `cachemiss`.

Generalizing from numerous tests like those just shown, we conclude that at the time scale required for the control of the active silicon temperature, the system is decoupled. Coupling is provided by the bulk, the metal connections, and the heat spreader, but all of these apparently act in the long run with respect to the scale mentioned above. As a consequence, the control system can be totally decentralized, and given the observed dynamics, can be composed of quite simple purely reactive SISO controllers. This makes our approach different from predictive or optimal control ones, like those quoted in Section II, and reliefs from the necessity of devising a prediction-targeted disturbance model—a difficult task

that may require articulated mathematics, as witnessed, e.g., by Bogdan [27].

Of course, we assume that if core temperatures as provided by the on-chip sensors stay within the prescribed limits, then no thermal problem exists at all. In other words, concerning the sizing and positioning of the sensors, hence the representativeness of their output as for the chip health, we cannot but trust the experience, thus the choices, of the manufacturers.

For completeness, one may want to make a distinction between two types of coupling that we name "physical" and "operational." Physical coupling is caused by thermal contact; its existence and entity—no matter how quantified—-are just a matter of fact. Operational coupling is caused by the activity of the processor, including intrachip communication, but also, for example, thread migration, and so forth. In fact one may question whether or not operational coupling is coupling in system theoretical terms, as it is obtained by coordinately manipulating *inputs* to the thermal system and does not reside in the system itself, however if we consider "controlled system" the running software as well, then discussing operational coupling is practically sound.

In the experiments of Fig. 1, there is neither core-to-core communication nor task migration or remote spawning, hence only physical coupling is evidenced. This is enough to justify our proposal of a controller per core, as the said proposal also entails a purely reactive approach, not caring whether the local load generating the experienced thermal stress comes from threads spawned locally or from another core. Of course applications can introduce operational coupling, but given again the reactive approach adopted, this makes no difference for the controller of a core, as it is just seen as an additional thermal stress for that core. The only relevant point in this respect is that operational coupling can be affected by the way the operating system allocates tasks to cores, which also influences communication. This is outside the scope of this paper, however, and correspondingly the proposed solution is transparent to task allocation and migration.

On a similar front, a malicious software cannot generate an unexpected power consumption pattern to cause a workload prediction to systematically fail, and thus endanger the system—again, because we do not rely on any workload prediction at all. Such an attack can thus provoke a performance degradation, but not jeopardize the chip safety.

The tests of Fig. 1 also show that the duration of fast thermal transients is dominated by a single time constant that does not appear to change in the various conditions examined. This is consistent with physics: uncertainty is concentrated in the way load and frequency turn into power—a phenomenon that is practically instantaneous. As such, we write our control-oriented model for one core in the form

$$C_c \frac{dT_c(t)}{dt} = g(\ell_c, T_c, t)\, f_c(t) - G_{cb}(T_c(t) - T_b(t)) \quad (1)$$

where C_c is the core heat capacity, G_{cb} is the equivalent core-bulk thermal conductance, f_c is the frequency, ℓ_c is the load, T_c is the core temperature, and T_b that of the bulk. The function $g(\ell_c, T_c, t)$ is the frequency-to-power gain: in principle it depends on the load, the temperature if leakage effects need accounting for, and directly on time, to represent the variability of the software behavior.

The measurable quantities are T_c and ℓ_c. However, by generating two different thermal stresses for the same load with `cpuburn` and `cachemiss`, we just showed that using ℓ_c as a proxy for power, which is what one really needs for control, can be very misleading. Estimating power requires information on the activity of individual units within a core. Some μPs contain "performance counters" from which the said information can be inferred, but doing so fast enough can be problematic, and in any case is architecture-specific. Also, the load-to-power gain is software-dependent, and thus it can change even faster than the fastest thermal dynamics. This significantly diminishes the importance of the dependence of $g(\cdot,\cdot,\cdot)$ on T_c. Finally, as $g(\cdot,\cdot,\cdot)$ depends on the short-time software behavior, its upper and lower bounds can be estimated by profiling the architecture, but its form is ultimately unknown. Hence, to synthesize the controller, we shall consider the simplified linear, time-varying (LTV) model

$$C_c \frac{dT_c(t)}{dt} = g(t)\, f_c(t) - G_{cb}(T_c(t) - T_b(t)) \quad (2)$$

where $T_b(t)$ will be treated as a (slow) additive load disturbance, and $g(t)$ as a time-varying gain, or equivalently a multiplicative input disturbance, with known upper and lower bounds. Incidentally, the use of LTV models for temperature control problems can be encountered in the literature (see the recent papers [28], [29]).

V. Control Synthesis

In this section, we first show that (2) can be stabilized—in the sense specified in the following—by a fixed-parameter PI, then introduce a procedure to tune the said PI, and finally structure the complete continuous-time control scheme.

A. Ensuring Stability

To compact the notation, we study the feedback system composed of the following.

1) The LTV first-order SISO process

$$\begin{cases} \dot{x}_P(t) = a_P x_P(t) + b_P(g(t)u(t) + d(t)) \\ y(t) = x_P(t) \end{cases} \quad (3)$$

with $a_P < 0$, $b_P > 0$, and $0 < g_1 \le g(t) \le g_2 < \infty \forall t$, where $u(t)$ is the control input, $y(t)$ is the controlled variable, and $d(t)$ is an additive disturbance.

2) The fixed-parameter PI controller

$$\begin{cases} \dot{x}_R(t) = b_R(w(t) - y(t)) \\ u(t) = x_R(t) + d_R(w(t) - y(t)) \end{cases} \quad (4)$$

where $w(t)$ is the set point.

The considered feedback system, in which the process model (3) corresponds to (2) by setting

$$u(t) = f_c(t), \quad x_P(t) = y(t) = T_c(t), \quad d(t) = G_{cb}T_b(t)$$
$$a_P = -G_{cb}/C_c \quad b_P = 1/C_c \quad (5)$$

is depicted in Fig. 2.

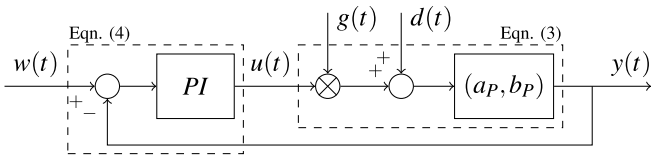

Fig. 2. Considered control loop.

Defining the state vector $x(t) = [x_P(t)\ x_R(t)]'$, the closed loop is ruled by

$$\dot{x}(t) = A(t)x(t) + b_w(t)w(t) + b_d d(t) \tag{6}$$

where

$$\begin{aligned} b_d &= [b_P \quad 0]' \\ A(t) &= \alpha(t)A_1 + (1-\alpha(t))A_2 \\ b_w(t) &= \alpha(t)b_{w1} + (1-\alpha(t))b_{w2} \end{aligned} \tag{7}$$

with

$$\begin{aligned} A_{\{1,2\}} &= \begin{bmatrix} a_P - b_P g_{\{1,2\}} d_R & b_P g_{\{1,2\}} \\ -b_R & 0 \end{bmatrix} \\ b_{w\{1,2\}} &= \begin{bmatrix} b_P g_{\{1,2\}} d_R \\ b_R \end{bmatrix}, \quad \alpha(t) = \frac{g(t) - g_1}{g_2 - g_1} \end{aligned} \tag{8}$$

hence $0 \leq \alpha \leq 1$. Our goal is to show that for the time-varying autonomous system $\dot{x}(t) = A(t)x(t)$, with matrix $A(t)$ as per (7), there exist PI parameterizations that make the origin of the state space a globally asymptotically stable equilibrium.

For any constant $\overline{\alpha}$, the corresponding constant matrix $\overline{A}$ is Hurwitz if and only if

$$b_R > 0, \quad d_R > \frac{a_P}{b_P \overline{g}}, \quad \overline{g} = g_1 + \overline{\alpha}(g_2 - g_1) \tag{9}$$

and, therefore—mind the sign of a_P—the set

$$\Pi_{[g_1,g2]} = \left\{ b_R > 0,\ d_R > \frac{a_P}{b_P g_2} \right\} \tag{10}$$

contains all the PI parameterizations that make both A_1 and A_2 Hurwitz. Ensured this, a sufficient condition for our claim to be true is that for some (b_R, d_R), A_1 and A_2 admit a common Lyapunov function.

In force of [30, Th. 3.1], the said matrices admit a common (quadratic) Lyapunov function if and only if A_1A_2 and $A_1A_2^{-1}$ have no real negative eigenvalues.

The eigenvalues of $A_1A_2^{-1}$ are 1 and g_1/g_2, both real positive, whereas the characteristic polynomial of A_1A_2 is

$$\begin{aligned} &p_{A_1A_2}(s) \\ &\quad = s^2 + \big(b_P(a_P d_R + b_R)(g_1 + g_2) - b_P^2 d_R^2 g_1 g_2 - a_P^2\big)s \\ &\qquad + b_P^2 b_R^2 g_1 g_2. \end{aligned} \tag{11}$$

Hence, a sufficient stability condition—adequate for us—is that $(b_R, d_R) \in \Pi_{[g_1,g2]}$, and both the roots of $p_{A_1A_2}(s)$ have a nonnegative real part, that is

$$b_P(a_P d_R + b_R)(g_1 + g_2) - b_P^2 d_R^2 g_1 g_2 - a_P^2 \leq 0. \tag{12}$$

Since the process (3), focusing on the input $u(t)$, can be viewed as a time-varying gain cascaded to a linear, time-invariant dynamics, an interesting particular case is when the PI is tuned by canceling the pole of the said dynamics, that is, when $-b_R/d_R = a_P$. In this case, condition (12) reduces to

$$-b_P^2 d_R^2 g_1 g_2 - a_P^2 \leq 0 \tag{13}$$

that is satisfied for any value of d_R.

B. Tuning the PI

Given the need for adaptability evidenced in the introductory part, we need a tuning procedure based on data that can be gathered after deployment. We chose to employ step response information like that shown in Fig. 1. The procedure outlined in this section could thus be automated to run for example at boot time, although we do not discuss the matter in this paper.

The procedure consists of keeping the core minimally loaded, which makes $g(t)$ almost constant and close to g_1, and applying a DVFS step in open loop, then leading the core to consume (almost) its maximum power, thereby making $g(t)$ approach g_2, and repeating the DVFS step test.

The two responses so obtained, recalling the dominantly first-order character of the controlled dynamics and the independence of its time constant of the operating point, easily yield the said time constant—τ_c to name it—and two extremal values $\mu_{c,\min}$ and $\mu_{c,\max}$ for the gain μ_c of a model in the form

$$P(s) := \frac{T_c(s)}{F_c(s)} = \frac{\mu_c}{1 + s\tau_c} \tag{14}$$

where interpreting based on (5)

$$\tau_c = \frac{C_c}{G_{cb}}, \quad \mu_{c,\min} = \frac{g_1}{G_{cb}}, \quad \mu_{c,\max} = \frac{g_2}{G_{cb}}. \tag{15}$$

We then tune (3) by cancelation, aiming at a dominant closed-loop time constant τ° and assuming a nominal value $\mu_{c,\text{nom}}$ for μ_c—for example, the center of the $[\mu_{c,\min}, \mu_{c,\max}]$ interval. This corresponds to requiring the open-loop transfer function

$$L(s) := \frac{\mu_{c,\text{nom}}}{1 + s\tau_c}\left(\frac{b_R}{s} + d_R\right) \tag{16}$$

to equal $1/s\tau^\circ$, and together with the fulfillment of the stability condition (13), is achieved by setting

$$d_R = \frac{\tau_c}{\mu_{c,\text{nom}}\tau^\circ}, \quad b_R = \frac{1}{\mu_{c,\text{nom}}\tau^\circ}. \tag{17}$$

It is worth noticing that if the controller was really built in the continuous time, one could make only the step test aimed at estimating $\mu_{c,\min}$, and use that value for the tuning without stability issues. This is not true when the event-based realization comes into play, however, whence the way we designed the tuning experiment. Also, rigorously speaking, tuning the PI this way does not provide guarantees on the efficacy of the system at rejecting the multiplicative disturbance $g(t)$, which is the primary control purpose, since the slow disturbance $d(t)$—yielded by the bulk dynamics—is eliminated by the integral action. Nonetheless, it is quite intuitive that widening the control band favors the required rejection. Further discussions on stability and performance are carried out directly on the event-based realization of the proposed control, in Section VI later on.

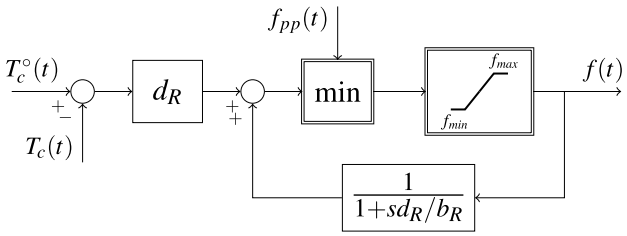

Fig. 3. Continuous-time control scheme for one core.

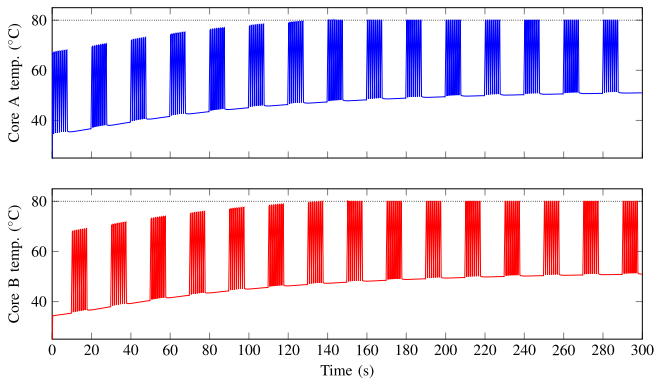

Fig. 4. Simulation test with the proposed scheme on two cores, in the continuous time.

C. Continuous-Time Control Scheme

As anticipated, we aim at integrating thermal control with power/performance management, and once again for adaptability, the former has to be agnostic on both the *rationale* of the latter, and how that *rationale* is realized. Given this, the control scheme we adopted is composed of the PI just synthesized that receives as set point a value "slightly lower" than the limit temperature—how much lower will be briefly discussed in Section VI-E, devoted to implementation.

Fig. 3 reports the scheme: $T_c^\circ(t)$ is the set point for the core temperature $T_c(t)$, $f(t)$ is the frequency command in the range $[f_{\min}, f_{\max}]$, and $f_{pp}(t)$ is the frequency requested by the power/performance manager, no matter what its policy is, see [31], [32] and the papers quoted therein to appreciate the variety of available solutions. As can be seen, integration with power/performance management is simply achieved via override: the lower frequency is selected between that requested by the power/performance management and that computed by the PI, the state of which behaves in accordance with the selection. Doing so results in a very simple structure that naturally makes the power/performance request prevail when the temperature is far enough below the limit, and thermal control take over in the opposite case.

Prior to the event-based realization, we tested the control scheme in the continuous time by means of a Modelica simulator of the chip, built with the library presented in [6]. Fig. 4 shows an example in which the temperature of two adjacent cores (Core A and Core B) is controlled to stay below 80 °C by their two PIs, while the two cores are alternately subjected to a highly variable load, or left idle.

The simulated power/performance governor is very simple, just allotting frequency proportionally to load, but this is inessential for the purpose of the test. As can be seen, in the frequency band of interest, decentralized control is absolutely adequate, while the coupling provided by bulk, spreader, and sink just results in a very low-frequency effect. We omit further details to leave room for experimental data later on.

VI. Event-Based Realization

We have established that the proposed decentralized control fits the problem, thus a fixed-rate realization of the proposed controller fits as well. The point is that the sampling rate for such a realization would be so high to result in an unacceptable computational overhead. To avoid this, we now introduce an event-based realization that for flexibility reasons endows each core with a hardware event generator and a software controller.

A. Hardware Event Generator

The event generator combines a send-on-delta and a timeout policy. Its operation is summarized by the flow diagram of Fig. 5. The core temperature is sampled at a fixed, small interval q_s (thus events are generated only at times that are integer multiples of q_s). If the new sample differs in magnitude by more than Δ from the sample when the controller was last run, the controller is invoked again, otherwise the sample is dropped. If, however, a timeout Θ expires since the last control computation, the controller is run unconditionally. To reduce noise-triggered events, it is convenient in practice to complete the event generator with a mild—say first order—lowpass filter on the sampled variable. This is easily realized with a register, a multiplier, and an adder, and we assume that neglecting the so introduced dynamics has no significant influences on the analysis carried out in the following—an assumption confirmed by the experiments. Fig. 5 does not represent the filtering of the sampled variable for simplicity.

The value of Θ is adapted to find a compromise between the contrasting needs for control quality and low overhead. If the controller was invoked due to a timeout, Θ is increased up to a maximum value $\Theta_{\max}$. If instead the controller ran for a temperature threshold event, Θ is reduced to q_s, which forces the controller to run again at the next temperature sample. In this case, however, the generator is then forced to wait for the timeout independently of possible temperature events, to avoid an excessive event frequency, see [33] for a discussion on this matter. The flag indicated with `waitTO` in Fig. 5 serves to this purpose.

The event generator can be realized in hardware as a quite simple state machine. To obtain area and consumption overhead estimates, we implemented and simulated the generator in RTL Verilog, and then synthesized it in Cadence Encounter with the NAND Gate Liberty standard cell library. Considering an operating voltage of 1.1 V and a clock frequency of 667 MHz, we obtained a per-core area and power overhead of 159 $(\mu\text{m})^2$ and 471 μW, respectively. Although detailed evaluations in terms of silicon space, added-value costs, and power overhead are architecture- and production-specific, thus not within the scope of this paper, we bear to state that the expected impact is quite low for any architecture.

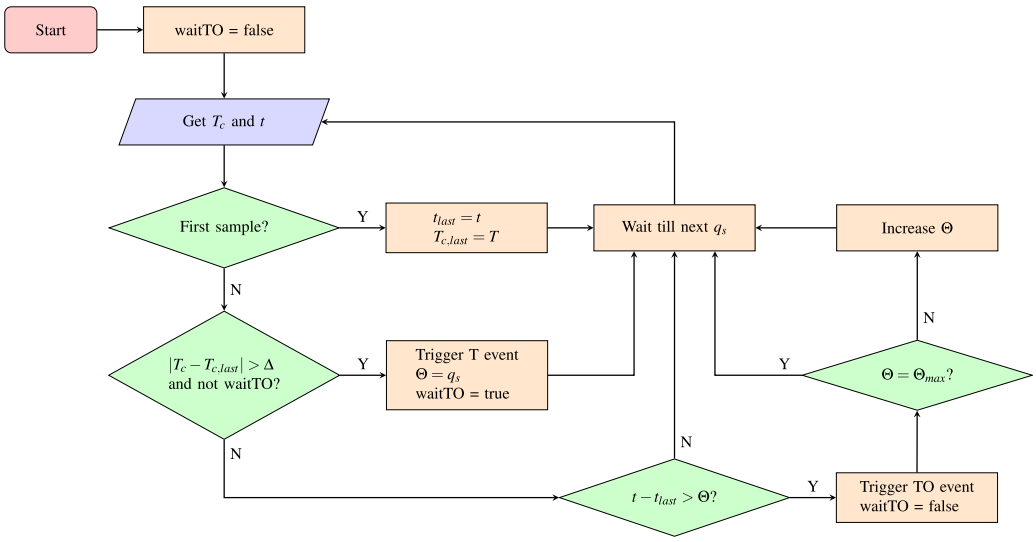

Fig. 5. Flow diagram for the hardware event generator's behavior.

B. Software Controller

For the purpose of this section, we represent the controlled system with a discrete-time model at timestep q_s, and assume that q_s is small enough to make any conclusion drawn using the said discrete-time model valid also for the real system, where the process is continuous time. To maximize the acceptability of this approximation, we employ the exponential discretization method, which is exact as the process input (the control signal) is constant on each period. Resorting for compactness to the notation introduced in (3) and (4), and distinguishing discrete-time quantities with a "*" superscript, we thus describe the process with

$$\begin{cases} x_P^*(h) = a_P^* x_P^*(h-1) + b_P^*(h) u^*(h-1) + \beta_P^* d^*(h-1) \\ y^*(h) = x_P^*(h) \end{cases} \tag{18}$$

where x_P^* is the state, while

$$\begin{aligned} a_P^* &= e^{a_P q_s}, \quad b_P^*(h) = \frac{b_P g(h q_s)}{a_P}(e^{a_P q_s} - 1) \\ \beta_P^* &= \frac{b_P}{a_P}(e^{a_P q_s} - 1) \end{aligned} \tag{19}$$

and the time index h is associated with the sampling at timestep q_s. Analogously, we write the discrete-time PI in state space form as

$$\begin{cases} x_R^*(k) = x_R^*(k-1) + b_R^*(w^*(k-1) - y^*(k-1)) \\ u^*(k) = x_R^*(k) + d_R(w^*(k) - y^*(k)). \end{cases} \tag{20}$$

The index k counts the control events, that owing to how the event generator is built, can only occur at multiples of h. When we need to locate the kth event in the sampling at step q_s, we shall indicate its h index with $h(k)$, whereas when this is not necessary we shall write, for the generic signal, $v(k)$ to indicate $v(h(k))$. Note also that (20) is a PI representation equivalent to Fig. 3 when no saturation occurs, which is what we need to study in this section.

We admit that the set point can be sensed only at events, which is reasonable as the limit temperature is seldom (if ever) modified, so that from the controller's viewpoint

$$w(l) = w(h(k-1)) = w(k-1), \quad l = h(k-1) \dots h(k) - 1 \tag{21}$$

and finally, since we are concerned with stability, we consider the case $d = 0$ for simplicity.

When the sensor triggers the kth event, it transmits to the controller the current *and the previous* samples of the controlled variable, i.e., $y^*(h(k))$ and $y^*(h(k)-1)$. The controller then computes $u^*(k)$ and holds it up to the subsequent event, hence

$$\begin{aligned} u^*(l) &= u^*(h(k-1)) \\ &= u^*(k-1), \quad l = h(k-1) \dots h(k) - 1. \end{aligned} \tag{22}$$

Prior to computing $u^*(k)$, x_R^* is made consistent with the received $y^*(h(k)-1)$, thus

$$\begin{aligned} x_R^*(h(k)-1) &= u^*(h(k)-1) \\ &\quad - d_R\big(w^*(h(k)-1) - x_P^*(h(k)-1)\big) \\ &= u^*(k-1) - d_R(w^*(k-1) - x_P^*(h(k)-1)) \end{aligned} \tag{23}$$

and, therefore

$$\begin{aligned} x_R^*(k) &= x_R^*(h(k)-1) + b_R^*\big(w^*(k-1) - x_P^*(h(k)-1)\big) \\ &= u^*(k-1) - d_R\big(w^*(k-1) - x_P^*(h(k)-1)\big) \\ &\quad + b_R^*\big(w^*(k-1) - x_P^*(h(k)-1)\big) \\ &= u^*(k-1) + \big(b_R^* - d_R\big)\big(w^*(k-1) - x_P^*(h(k)-1)\big) \end{aligned} \tag{24}$$

from which, since

$$u^*(k-1) = x_R^*(k-1) + d_R\big(w^*(k-1) - x_P^*(k-1)\big) \tag{25}$$

we get

$$\begin{aligned} x_R^*(k) &= x_R^*(k-1) + d_R\big(w^*(k-1) - x_P^*(k-1)\big) \\ &\quad +(b_R^* - d_R)w^*(k-1) \\ &\quad -(b_R^* - d_R)x_P(h(k)-1) \\ &= \dots \\ &= x_R^*(k-1) + b_R^* w^*(k-1) - d_R x_P^*(k-1) \\ &\quad -(b_R^* - d_R)x_P^*(h(k)-1). \end{aligned} \tag{26}$$

If we now evidence the variation of x_P^* from the $(k-1)$th event until "immediately" (i.e., q_s) before the k-to one by writing

$$x_P^*(h(k)-1) = x_P^*(k-1) + \delta x_P^*(k-1) \tag{27}$$

where the index $k-1$ attributed to δx_P^* is sensible, because that quantity is known before the kth event is triggered, we obtain

$$\begin{aligned} x_R^*(k) &= x_R^*(k-1) - b_R^* x_P^*(k-1) + b_R^* w^*(k-1) \\ &\quad +(d_R - b_R^*)\delta x_P^*(k-1). \end{aligned} \tag{28}$$

Reasoning in a similar way for the controlled system's state, we have

$$\begin{aligned} x_P^*(k) &= a_P^* x_P^*(h(k)-1) + b_P^*(h(k))u^*(h(k)-1) \\ &\quad a_P^* x_P^*(h(k)-1) + b_P^*(k)u^*(k-1) \\ &\quad a_P^*(x_P^*(k-1) + \delta x_P^*(k-1)) + b_P^*(k)u^*(k-1) \\ &= \dots \\ &= \big(a_P^* - b_P^*(k)d_R\big)x_P^*(k-1) + b_P^*(k)x_R^*(k-1) \\ &\quad + b_P^*(k)d_R w^*(k-1) + a_P^*\delta x_P^*(k-1). \end{aligned} \tag{29}$$

Putting it all together, we can describe the closed-loop system in the k index—i.e., counting the events irrespectively of their distance in the constant-rate sampled time with step q_s—as

$$\begin{cases} x^*(k) = A^*(k)x^*(k-1) + b^*(k)w^*(k-1) + f^*\delta x_P^*(k-1) \\ o^*(k) = C^* x^*(k) + D^* w^*(k) \end{cases}$$

where $x^*(k) := [x_P^*(k)\, x_R^*(k)]'$, $o^*(k) := [y^*(k)\, u^*(k)]'$ and

$$A^*(k) = \begin{bmatrix} a_P^* - b_P^*(k)d_R & b_P^*(k) \\ -b_R^* & 1 \end{bmatrix}, \quad b^*(k) = \begin{bmatrix} b_P^*(k)d_R \\ b_R^* \end{bmatrix}$$
$$f^* = \begin{bmatrix} a_P^* \\ d_R - b_R^* \end{bmatrix} \quad C^* = \begin{bmatrix} 1 & 0 \\ -d_R & 1 \end{bmatrix}, \quad D^* = \begin{bmatrix} 0 \\ d_R \end{bmatrix}. \tag{30}$$

C. Event-Based Stability

As we did for the continuous-time scheme in Section V-A, we now show that for the time-varying autonomous system $x^*(k) = A^*(k)x^*(k-1)$, with matrix $A^*(k)$ as per (30), there exist parameterizations for the event-based PI just described that make the origin of the state space a globally asymptotically stable equilibrium.

Recalling (19), as $g(h(k)q_s)$ varies in the $[g_1, g_2]$ interval, $b_P^*(k)$ will vary in a $[b_{P1}^*, b_{P2}^*]$ one, hence setting

$$A_1^* = \begin{bmatrix} a_P^* - b_{P1}^* d_R & b_{P1}^* \\ -b_R^* & 1 \end{bmatrix}, \quad A_2^* = \begin{bmatrix} a_P^* - b_{P2}^* d_R & b_{P2}^* \\ -b_R^* & 1 \end{bmatrix}. \tag{31}$$

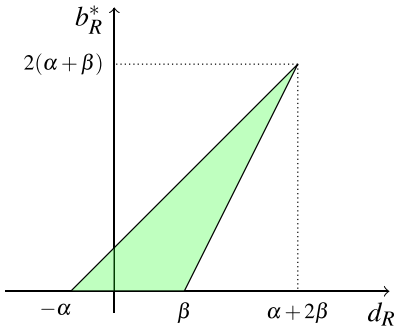

Fig. 6. Region in the (d_R, b_R^*) plane that makes both A_1^* and A_2^* Schur.

We can write

$$A^*(k) = \gamma(k)A_1^* + (1-\gamma(k))A_2^*, \quad 0 \le \gamma(k) \le 1 \quad \forall k. \tag{32}$$

Omitting lengthy but trivial computations based on the Jury criterion, the set of (d_R, b_R^*) couples that make both A_1^* and A_2^* Schur is the region internal to the triangle shown in Fig. 6, where

$$\alpha = \frac{1 - a_P^*}{b_{P2}^*}, \quad \beta = \frac{1 + a_P^*}{b_{P2}^*}. \tag{33}$$

Once ensured that A_1^* and A_2^* are Schur, to prove our claim we need to prove that for some (d_R, b_R^*), they admit a common Lyapunov function. According to the result in [34, Sec. 4], here reported with some notation modifications to avoid confusion, A_1^* and A_2^* admit a common (quadratic) Lyapunov function if and only if, defining the two matrices

$$\begin{aligned} P^* &= \big[p_{ij}^*\big] := 0.5I - \big(I + A_1^*\big)^{-1} \\ Q^* &= \big[q_{ij}^*\big] := 0.5I - \big(I + A_2^*\big)^{-1} \end{aligned} \tag{34}$$

and computing the quantities

$$\begin{aligned} r_1 &:= p_{11}^* q_{22}^* + p_{22}^* q_{11}^* - p_{12}^* q_{21}^* - p_{21}^* q_{12}^* \\ r_2 &:= \det(Q^*) \\ r_3 &:= r_1 - 2r_2 \\ r_4 &:= \det(P^*) + r_2 - r_1 \end{aligned} \tag{35}$$

the following conditions hold true.

1) $r_4 \le 0$, or
   $r_4 > 0$, $-r_3/2r_4 \notin [0,1]$, or
   $r_4 > 0$, $-r_3/2r_4 \in [0,1]$, $r_2 - r_3^2/4r_4 > 0$.
2) The same as above replacing A_2^* in Q^* with $-A_2^*$.

With the (A_1^*, A_2^*) couple, we obtain $r_4 = 0$; for the $(A_1^*, -A_2^*)$ couple, things are more complex, and for convenience, we bring in the cancelation-based tuning policy introduced previously. In the discrete time—recall (18) and (20)—this policy means setting $b_R^* = (1 - a_P^*)d_R$, which yields the closed-loop complementary sensitivity function

$$\frac{T_c(z)}{T_c{}^\circ(z)} = \frac{b_P^* d_R}{z - 1 + b_P^* d_R}. \tag{36}$$

We already chose $d_R > 0$ based on (10), but since it is sensible to have the closed-loop pole in the range $(0,1)$ to avoid oscillations, we introduce the further condition

$$0 < 1 - b_P^* d_R < 1 \quad \forall b_P^* \in \big[b_{P1}^*, b_{P2}^*\big] \tag{37}$$

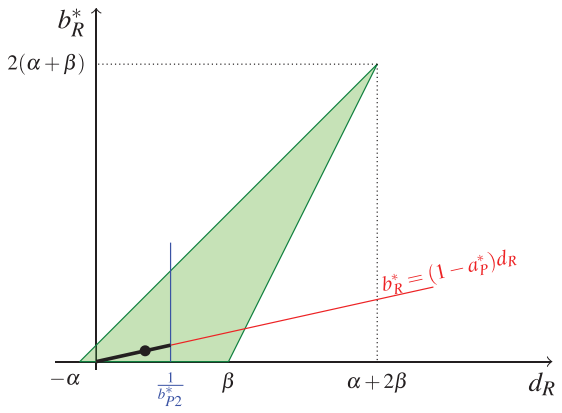

Fig. 7. Event-based stability region for cancelation-based tuning.

whence

$$0 < d_R < \frac{1}{b_{P2}^*}. \tag{38}$$

Setting $b_R^* = (1-a_P^*)d_R$, for the $(A_1^*, -A_2^*)$ couple, we get

$$r_4 = \frac{2a_P^*}{(1-a_P^{*2})b_{P2}^*} \frac{(b_{P1}^* + b_{P2}^*)d_R - 2}{d_R(b_{P1}^* d_R - 2)} \tag{39}$$

that is negative, for $d_R > 0$, if

$$\frac{2}{b_{P1}^* + b_{P2}^*} < d_R < \frac{2}{b_{P1}^*}. \tag{40}$$

The expression of $-r_3/2r_4$ is more complex and omitted, except for noticing the following.

1) It is defined and continuous for $d_R \neq 2/(b_{P1}^* + b_{P2}^*)$.
2) Its limit for $d_R \to 0^+$ is

$$\frac{a_P^{*2} + 6a_P^* + 1}{8a_P^*}$$

which for $0 < a_P^* < 1$ is greater than the unity.
3) Its derivative with d_R is positive for $0 < d_R < 4/(b_{P1}^* + b_{P2}^*)$.

Since $b_{P2}^* > b_{P1}^* > 0$ implies $2/(b_{P1}^* + b_{P2}^*) > 1/b_{P2}^*$, in the d_R range of interest either $r_4 \leq 0$, or if this is not true, $-r_3/2r_4 > 1$ anyway. This proves the claim.

The obtained result is shown graphically in Fig. 7, where the used stability region is evidenced by the thick black segment.

D. Tuning the Event-Based PI

The event-based PI is tuned from the measured μ_c, τ_c, and from the specification τ°—see (17)—by setting

$$d_R = \frac{\tau_c}{\mu_c \tau^\circ}, \quad b_R^* = \frac{\tau_c}{\mu_c \tau^\circ}(1 - e^{-q_s/\tau_c}). \tag{41}$$

Based on the tuning experiment described in Section V-B, for the used μP, we have

$$\begin{aligned} \tau_c &= 20 \text{ ms} \\ \mu_{c,\min} &= \frac{11 \text{ °C}}{4.2 \text{ GHz} - 0.96 \text{ GHz}} = 3.4 \text{ °C/GHz} \\ \mu_{c,\max} &= \frac{21 \text{ °C}}{4.2 \text{ GHz} - 0.96 \text{ GHz}} = 6.5 \text{ °C/GHz}. \end{aligned} \tag{42}$$

Widening the μ_c span by 20% for safety, and taking $\mu_{c,\text{nom}}$ as the said span's midpoint, we get

$$\mu_{c,\min} = 2.7 \text{ °C/GHz}, \quad \mu_{c,\max} = 7.8 \text{ °C/GHz} \tag{43}$$

$$\mu_{c,\text{nom}} = 5.25 \text{ °C/GHz} \tag{44}$$

that with $q_s = 5$ ms and $\tau^\circ = 10$ ms give

$$\begin{aligned} a_P^* &= e^{-q_s/\tau_c} = e^{-5 \text{ ms}/20 \text{ ms}} \\ &= 0.78 \\ b_{p1}^* &= \mu_{c,\min}(1 - a_P^*) = 2.7 \text{ °C/GHz} \cdot (1 - 0.78) \\ &= 0.59 \text{ °C/GHz} \\ b_{p2}^* &= \mu_{c,\max}(1 - a_P^*) = 7.5 \text{ °C/GHz} \cdot (1 - 0.78) \\ &= 1.72 \text{ °C/GHz} \\ \alpha &= (1 - a_P^*)/b_{p2}^* = (1 - 0.78)/(1.72 \text{ °C/GHz}) \\ &= 0.13 \text{ GHz/°C} \\ \beta &= (1 + a_P^*)/b_{p2}^* = (1 + 0.78)/(1.72 \text{°C/GHz}) \\ &= 1.03 \text{ GHz/°C} \\ 1/b_{p2}^* &= 1/(1.72 \text{ °C/GHz}) \\ &= 0.58 \text{ GHz/°C} \\ d_R &= \tau_c/(\mu_{c,\text{nom}} \tau^\circ) = 20 \text{ ms}/(5.25 \text{ °C/GHz} \cdot 10 \text{ ms}) \\ &= 0.38 \text{ GHz/°C} \\ b_R^* &= (1 - a_P^*)d_R = (1 - 0.78) \cdot 0.38 \text{ GHz/°C} \\ &= 0.08 \text{ GHz/°C}. \end{aligned} \tag{45}$$

The controller tuning in (45), marked by the black dot in Fig. 7 and well within the stability region, is the one used for the experimental tests reported in Section VII.

E. Performance

Among the various possible indicators for control performance, in this paper, we choose the closed-loop settling time. Supposing that the continuous-time PI tuned as per (17) provides satisfactory performance from this viewpoint, it is necessary to ensure that this satisfactory behavior carries over to the event-based realization. This requires to choose the send-on-delta threshold Δ properly. The point is that determining how many k steps—i.e., events—it takes for the free motion of (30) to converge to zero within a given tolerance is straightforward, but this does not give information about the same duration in time, as (30) does not account for the interval between two events.

Denoting by $\overline{N}_c$ the number of steps that the free motion of (30) takes to converge, a worst case estimation of the closed-loop settling time is obtained by multiplying $\overline{N}_c$ by the duration of the prescribed timeout. In general, however, such an estimate is too pessimistic. In fact it *has* to be pessimistic, because if the majority of the events triggered during a transient are caused by timeouts, excluding by hypothesis an incorrectly tuned PI, the send-on-delta threshold is definitely not adequate.

On the other hand, an optimistic estimate of the same settling time can be obtained by assuming that the number of h steps between each $(k-1)$th and kth event is equal to the send-on-delta threshold divided by the variation of y

over one h step, computed at the h step corresponding to the $(k-1)$th event, and rounded to the nearest integer. Summing the so approximated inter-event periods up to the $\overline{N}_c$ steps to convergence provides the settling time estimate.

We tested the two estimates mentioned above in a large number of experimental cases, and found that they can sometimes be so far from one another, to be of no practical use—that is why we do not delve into further details on the possible heuristics to obtain Δ this way. Hence, although we are convinced that the sought information on the settling time can be obtained by the controlled system and the event-based PI parameterization, there is still work to do in this direction. For the moment, Δ is selected by means of a design space exploration based on the tuning model (14), parameterized from the DVFS step response, which proved a successful enough technique in practice—see Section VII-B through Section VII-D later on.

The same exploration allows to determine how much lower the set point has to be with respect to the limit temperature, as long as this limit has to be taken as a hard one (which is not exactly the spirit of thermal management, however). Based on experience with a large number of tests, we decided that a temperature set point lower than the limit by about 1.5Δ is a safe choice. For completeness, it is also worth noticing that the selection of Δ is at present complicated by the low resolution of typical on-chip sensors, as it is apparently meaningless to set the threshold lower than the said resolution; however, discussing these technological aspects would stray from the scope of this paper.

Coming to the control overhead, we quantified it also for the software controller. We benchmarked the code with `RDTSCP` [35] instructions, and obtained an average of 40 clock cycles per step on the μP we used for the tests. At 2.4 GHz, this means that the time required to run the controller is 16 ns (neglecting cache misses that, however, have a limited effect given the minimal memory footprint); this places the proposed solution among the fastest ones. For comparison, [14] claims that a C implementation of their proposed control algorithm requires 7 μs. To further reduce its overhead, the control algorithm can be realized with fixed point arithmetic, allowing for a straightforward implementation in environments without floating point, such as the Linux kernel. The controller can also be made an interrupt service routine local to the core, avoiding high-speed intercore communication (a significant problem of centralized policies).

In addition to the PI parameters and to Δ, the event-based realization requires to select q_s and $\Theta_{\max}$. Parameter q_s represents the fastest reaction time to a thermal event, and can be selected as one would choose a "reasonable but tendentiously small" sampling time for a fixed-rate control realization. This decision is easily taken during the (online) tuning procedure, based on the model parameters and the desired response speed. In the case considered herein, a value of 5 ms is suitable.

As for $\Theta_{\max}$, if one cares only about thermal control there is no limit, as the send-on-delta mechanism does not depend on it; in [33], a general discussion on the matter can be found. However, since our scheme is conceived to integrate with power/performance, it is sensible to set the timeout to the desired reaction time to load changes. In the case addressed herein we set $\Theta_{\max}$ to 100 ms, so as to allow for very low event rates in the absence of thermal stress while reacting to load fast enough.

As said above for performance, there is still work to do also about robustness. In this respect, the simplicity of the used model becomes in some sense a problem, because the uncertainty caused, e.g., by product and installation variability may not be parametric, and even hard to describe in a structured manner. What we could do at present to figure out some "practical" robustness quantification, was to artificially detune the controller and observe the results. We found that even quite significant alterations of the controller parameters, although within the nominal stability region, still yield stable behaviors of the control system.

As a final remark, if the proposed hardware/software partition is adopted, the control reaction promptness is dictated by hardware, hence sufficient by orders of magnitude [36], and if the software generating the action is light enough, we do not expect significant timing issues also in the future. In the worst possible case, one could give up some configurability and make the control computation hardware as well.

VII. Experimental Results and Comparisons

This section first describes the experimental setup used to assess the proposed control, and compare it to the state of the art. The matter is worth attention because mainstream solutions differ from ours significantly, see Section III, hence ensuring a fair and meaningful comparison is not trivial. Then, selected experimental tests are presented and discussed.

A. Experimental Setup

We are presenting a complete (hardware plus software) control solution for integrated thermal and power/performance management. For testing, this poses two problems. First, we need complete authority over the chip, without influences of any preexisting thermal and power management. Second, we cannot (at present) realize the state machine in hardware.

Concerning the first issue, the mentioned influences come from the operating system and the hardware itself. As for the operating system, we chose Linux for manageability issues (replicating our results with other operating systems is possible if deep enough access to the system is available). To disable the default power/performance policy, we removed the `intel_pstate` module and selected the `userspace` Linux governor, which allows to set the clock frequency directly.

As for hardware, in the following, we refer to the processor used here (an Intel core i5-6600K), but considerations can be easily generalized to other devices. We left the thermal *protection* in place and operating, since it intervenes only at 100 °C, and thermal management should seldom (if ever) let a core reach that temperature. Another important issue to care about is the Turbo Boost that can set the clock frequency to values higher than those permitted to the operating system. Viewed from our thermal controller perspective, the Turbo

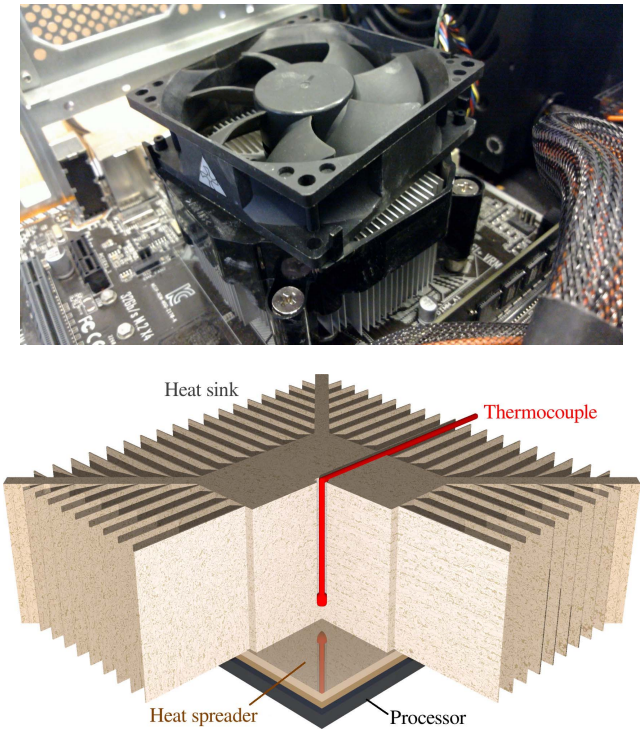

Fig. 8. Photograph of the processor with heat sink and fan (top) and positioning of the heat sink thermocouple (bottom).

Boost is a disturbance, so we had to disable it. Doing so does not alter the comparison *arena*, because the Turbo Boost is software transparent. However, with the Turbo Boost disabled, it is difficult to reach temperatures high enough to really stress the compared controls. We, therefore, overclocked the machine up to 4.2 GHz, i.e., slightly more than the 3.9 GHz achievable with the Turbo Boost. The *rationale* of our choice is to subject the compared controllers to reasonably harsh situations, without, however, causing an excessively severe thermal stress, that they would never encounter in any real-world installation.

In the experimental setup, see Fig. 8, the i5-6600K is mounted on an ASUS Z170K motherboard, and powered by a Corsair V6 650 supply. The power supply is deliberately over-sized, to prevent possible abrupt current jolts from affecting the results. The used temperature sensors are those aboard the chip, which we accessed via MSRs[2] as the operating system infrastructures are too slow. In addition, we inserted a thermocouple in the heat sink by drilling a hole down to about 3 mm from the side attached to the processor.

During the test we left in place the BIOS fan management, that with minor variations from one computer model to another operates to control the "package" temperature, wherever the sensor that measures the said temperature is located. The key point to have in mind for this test is that any action of the fan affects the active silicon only by traversing the thermal dynamics of the heat sink.

[2]Model Specific Registers, an Intel x86 feature for low-level fast access to processor features; similar mechanisms, with variations inessential herein, are provided by other manufacturers.

Coming to the second issue, as we could not realize the state machine in hardware, we emulated it with a fast task clocked at period q_s. The resulting overhead is very low and in any case disfavors our technique, so in the following, we just neglect it. Also, though for best operation the software controller should be a kernel thread, we decided to run it in user space for simplicity.

In the following, we present three experimental tests, each representative of a series of analogous ones, with the objectives summarized in the following.

1) Demonstrate, as far as permitted by the available instrumentation, the presence of the multiple time scales envisaged in Section II, and the ability of the proposed solution to control the relevant dynamics.
2) Compare the proposed solution to a classical and widespread one, where thermal issues are dealt with by hardware protections, and the power/performance tradeoff is managed by the operating system.
3) Compare the proposed solution to an advanced one, namely, the recently introduced Linux thermal daemon `thermald` [37].

As a final remark, since the i5-6600K has one DVFS for all its four cores, we configured our control to keep the *hottest* core temperature below the limit, which makes the assessment of the decoupled nature of the system somehow indirect. Experimenting with multiple DVFS inputs is among future research directions.

B. Experiment 1—Multiscale Dynamics

This experiment is performed by having the μP under our control while executing a benchmark taken from the Intel optimized `LINPACK` parallel suite [38]. The used benchmark consists in solving a system of linear equations. We used the binaries optimized with the Intel compiler, because they are maximally efficient, and, therefore, maximally stress the architecture from the thermal standpoint.

Fig. 9 reports the results. The top plot shows that none of the four cores' temperatures exceeds the limit that was set to 80 °C—in fact by zooming one can see just a few samples sparingly above the limit, but this is irrelevant for the sense the limit itself has in the addressed problem.

It is also possible to appreciate two of the envisaged time scales, namely, the fastest, captured by core sensors very near to the active silicon, and the slowest, represented by the heat sink. There is no sensor to actually view the intermediate-scale thermal phenomena relative to the bulk, but we can state that the observed results do support the modeling hypotheses of Section II, and the consequent choices as for control. In particular, from this test, it clearly appears that no control action having to traverse the sink dynamics can be of any use to contrast the fast thermal stress produced by the running code, whereas the proposed controller successfully governs the relevant dynamics.

This important fact can be further appreciated by noticing the evident high-frequency content in the control signal, see the bottom plot in Fig. 9. Thanks to the event-based realization, however, on average only 48.7 control actions per second are

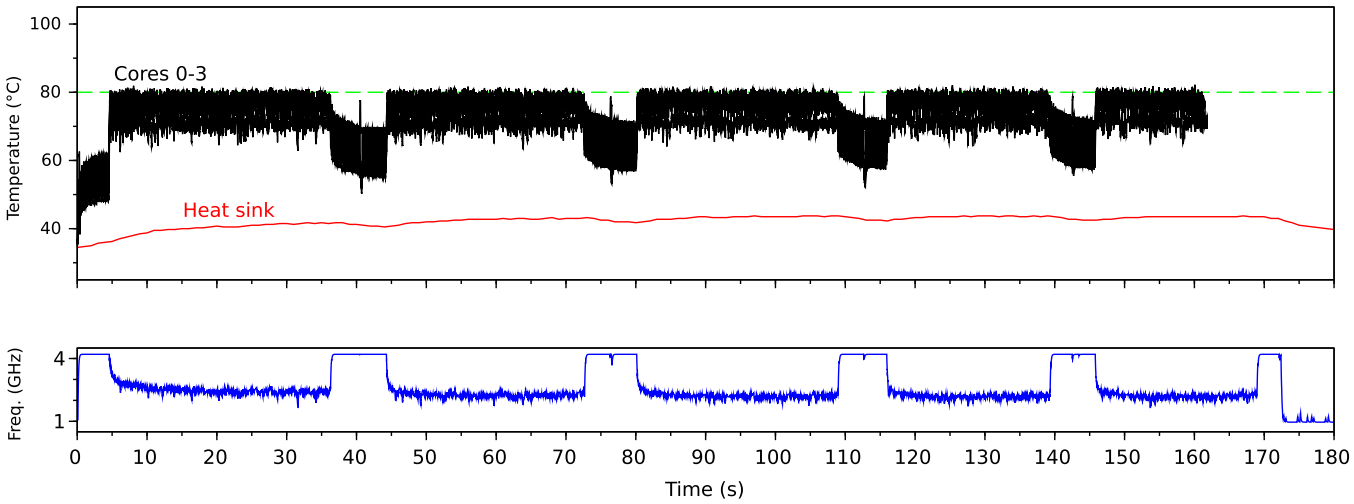

Fig. 9. Experimental test 1. Top: cores' and heat sink temperatures. Bottom: frequency.

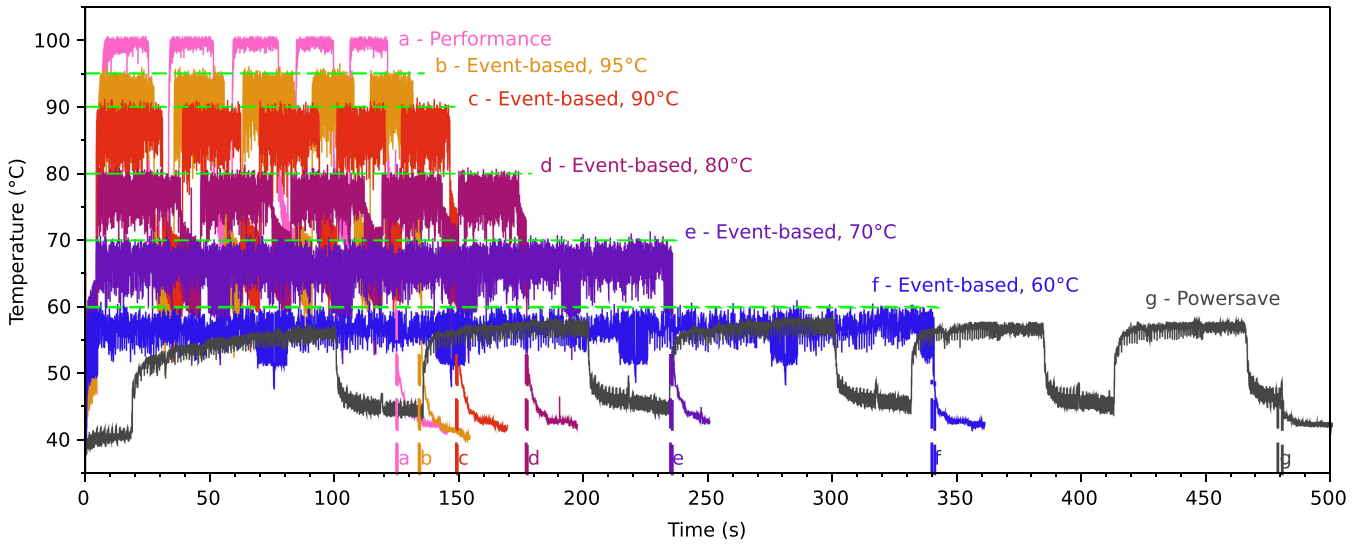

Fig. 10. Experimental test 2—temperature of the hottest core with the Linux governors `performance` and `powersave` (plots a and g, respectively) and with the proposed controller and different limit temperatures (plots b through f). The corresponding benchmark completion times are marked by the vertical dashed lines.

required over the benchmark duration. The same event rate, with a fixed-timestep controller, would require a sampling time of around 20 ms: with a dominant controlled system time constant of 20 ms, and the necessity of rejecting disturbances quickly, such a sampling time is simply out of question.

Finally, notice that the frequency reaches the upper saturation when the controlled temperatures go down. This happens because the benchmark keeps the processor 100% loaded throughout its execution, but alternates phases with many computations—thus consuming much power—to phases where the main activity is accessing memory, which is less power intensive. This supports our statement in Section IV, that using load as a proxy for power is potentially misleading, and, therefore, backs up our proposal as for the way to integrate thermal with power/performance management.

C. Experiment 2—Comparison With Linux Governors

This experiment is conducted with the same LINPACK benchmark used in Section VII-B. The counterpart is provided by a very typical Linux machine setup, where thermal management is delegated to the hardware protections, and the power/performance tradeoff is managed by a governor chosen among `performance`, that sets the frequency constant to the maximum, `powersave`, that sets it constant to the minimum, and `ondemand`, that adapts the frequency to the load based on some heuristics inessential to discuss herein.

The benchmark is run to completion with the `performance` and the `powersave` governors, and with the proposed event-based integrated controller. We also tested the `ondemand` governor that yields practically the same result of `performance` if not for a reduction of the frequency only when the benchmark terminates. The outcome of `ondemand` is thus omitted.

Fig. 10 shows the obtained results, and some synthetic figures are given in Table I. In the figure, only the temperature of the hottest core is plotted, so as to improve readability. The frequency is not plotted for the same reason, but the table gives its average value. The `performance` governor achieves the fastest completion, but at the cost of making the hardware protection intervene several times (in the figure, this is when

TABLE I

COMPARISON WITH LINUX GOVERNORS—AVERAGE EVENT RATES WHERE APPLICABLE, COMPLETION TIMES, AVERAGE FREQUENCIES, AND SLOWDOWN WITH RESPECT TO THE FASTEST COMPLETION BY `Performance`

Control	avg. ev/s	compl. time	avg. freq.	slowdown
Performance	n/a	124s	3.79GHz	
Ev. based, 95°C	50	135s	3.24GHz	9%
Ev. based, 90°C	54	149s	3.12GHz	20%
Ev. based, 80°C	57	177s	2.64GHz	42%
Ev. based, 70°C	64	235s	1.95GHz	89%
Ev. based, 60°C	22	340s	1.36GHz	2.7×
Powersave	n/a	479s	0.96GHz	3.9×

temperature is stuck around 100 °C). The `powersave` one, conversely, yields the lowest temperature but the longest completion. The average frequency with `performance` is not the maximum (4.2 GHz) owing to an action of the hardware thermal protection that is evident in the figure.

The point in this test is that by acting on the limit temperature, the proposed controller allows to move with continuity between the two extremal behaviors of the `performance` and the `powersave` governors. This can be appreciated from both the data in Table I and the plots in Fig. 10; we could not perform an event-based test at 100 °C, because it would be impossible to distinguish the action of the controller from that of the hardware thermal protection. In any case, as the limit temperature is increased, the slowdown with respect to `performance` approaches zero, indicating that the proposed control does not unduly limit computational speed. As for the average event rates, once again they apparently could not be achieved with a fixed-timestep controller.

The outcome of this test not only further corroborates the proposed integration of thermal and power/performance management, but indicates that the latter can be somehow obtained as a consequence of the former. Although further study is required, one can in fact conjecture that a convenient scheduling of the limit temperature can be used as a means to drive a computing system toward a "powerful" or an "energy saving" behavior, all with a single—and simple—controller like the one proposed herein.

D. Experiment 3—Comparison With the Linux Thermal Daemon

We finally compare the proposed controller to `thermald`, a recently introduced solution for thermal management, now available by default in several Linux distributions, that comprehends a PID controller, and is a good representative of the present state of the art.

The test shown here is done with a more articulated benchmark, which alternates two `LINPACK` executions with `cpuburn`, so as to provide a diverse load—and especially power—profile. The test is performed with a standard Linux configuration, including the `intel_pstate` module and the default governor; the temperature limit is set to 80 °C.

To compare the thermal behaviors, in this test, we use a penalty function J given by the integral of the squared temperature error when above the limit, that is

$$J = \int_{t=0}^{t_{\text{fin}}} f(T(t))dt \tag{46}$$

with

$$f(T) = \begin{cases} (T - T_{\text{lim}})^2 & T > T_{\text{lim}} \\ 0 & T \leq T_{\text{lim}} \end{cases} \tag{47}$$

where T is the controlled temperature, T_{lim} is the limit, and $[0, t_{\text{fin}}]$ is the experiment timespan. The *rationale* is to penalize a short but large limit violation more than staying for a longer time just above the limit. This choice is sensible if one considers the chip stress that follows in the long run from repeating many times either type of temperature limit violation, see e.g. [39] or the comprehensive work [40] and some papers quoted therein.

TABLE II

COMPARISON WITH `Thermald`—AVERAGE EVENT RATES WHERE APPLICABLE, COMPLETION TIMES, SLOWDOWN, AND VALUE OF THE PENALTY FUNCTION J AS PER (46)

Control	avg. ev/s	compl. time	slowdown	J
`thermald`	n/a	454s		$4.36{\cdot}10^4$°C^{2}s
Event-based	30	479s	5.5%	2.49°C^{2}s

Fig. 11 shows the obtained results, and some numbers are given in Table II. In the figure, it is possible to distinguish the `LINPACK` phases, with their high software-induced power variability, and the `cpuburn` ones—around 200 s and at the end of the test—where power is conversely almost constant if frequency is.

The first thing that emerges is that the completion time with `thermald` is lower, but only at the cost of a very significant degradation in the thermal behavior with respect to the proposed control. It is, however, particularly interesting to observe in more detail the control signals (center and lower plots). Contrary to the proposed controller, `thermald` acts with a time scale that is tendentiously too slow for the dynamics to govern, except for just a couple of small periods around 130 and 420 s (the high-frequency control activity observed when the temperature approaches 100 °C is produced by the hardware protection).

Hence, in this test, the proposed controller does exhibit an advance over the state of the art, but from a research viewpoint, this is not the very key point. The main lesson learned is the real necessity of making thermal management as independent as possible of the operating system, especially when the controlled hardware requires fast and timely action.

Indeed, the difficulties experienced by `thermald` come basically from the various preexisting operating system components involved in its operation that were not designed having the said involvement in mind.[3] As a qualitative proof for this statement, observing the bottom plot in Fig. 11, one notices a

[3]There is not the space for a complete description, but at https://01.org/linux-thermal-daemon/documentation/introduction-thermal-daemon the interested reader can find a block diagram, and the mention of some of the involved interfaces and modules like /sys/devices/platform/coretemp.x, /sys/class/thermal/, p_state and so on.

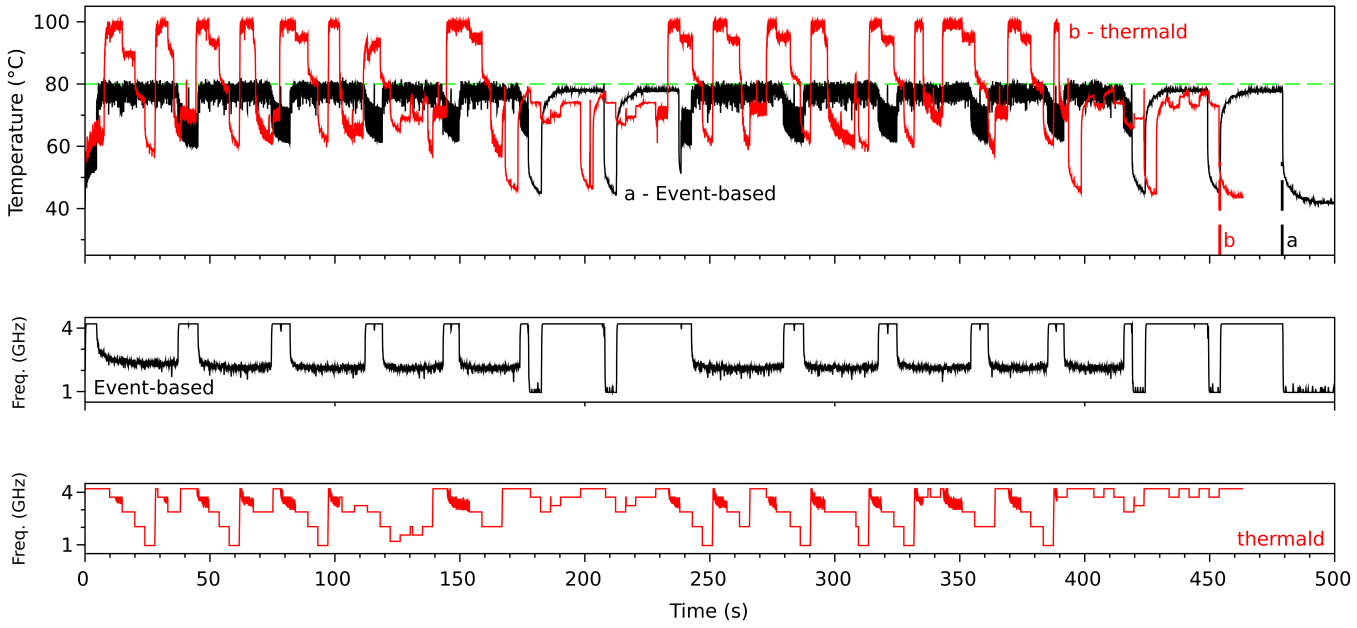

Fig. 11. Experimental test 3—temperature of the hottest core with proposed controller (a) and `thermald` (b) (top) and frequency with the two controllers (center and bottom); in the top plot, the two benchmark completion times are marked by the vertical dashed lines.

very oscillatory control behavior also in the `cpuburn` phases, i.e., a poor degree of stability. A complete analysis is not possible due to the difficulty of modeling `thermald` as a dynamic system. However, most likely a reason for the lack of stability is that the control action decided by the daemon has to traverse components that operate each with its own timing, and thus introduce a potentially significant and variable delay.

Summarizing, on one hand, this test supports the statement of Section II that a good solution has to be transparent and agnostic as for the operating system, and on the other hand, backs up the idea of splitting the solution into a hardware part to promptly call for action with guaranteed timing, and a software part to make that action as configurable as needed.

VIII. Conclusion and Future Work

In this paper, we presented a thermal controller for high-density microprocessors that integrates seamlessly with any power/performance management policy, and if simplicity is desired, can itself handle the power/performance tradeoff by acting on the limit temperature.

We motivated our approach by analyzing the history and the trends of the processor industry with respect to the treated problem, showing that nowadays the said problem is on the one hand vital for the chips, and on the other hand very keen to unduly limit computational speed if not properly addressed.

We outlined the characteristics that an effective solution has to possess in terms of performance, computational lightness, configurability, and independence with respect to the operating system.

We analyzed the controlled system, based on open-loop experiments and first-principle modeling, and concluded that a decentralized control is adequate. We set up the control in the continuous time, proved its stability accounting for its LTV nature, and verified in simulation the efficacy of the decentralized structure.

We showed that given the requirements as previously stated, an event-based realization is highly beneficial (not to say necessary) by determining the sampling time that a fixed rate one would require for an acceptable performance.

We designed such a realization, conveniently partitioning it into a hardware event generator and a software controller—an idea that we think may be fruitfully applied to other computer-related control problems, incidentally. The proposed structure was motivated as well suited to provide at the same time the necessary promptness and configurability.

We proved stability also for the event-based realization, carried out a preliminary performance analysis, and provided configuration guidelines.

We implemented the presented solution on real hardware, and showed by convenient testing that it fulfills the requirements, and yields visible advantages over the state of the art. In particular, we verified that our solution can play both the role of a thermal controller and of a power/performance manager, in a more flexible manner than the present governors. We also proved that the necessary promptness cannot be achieved without bypassing the operating system modules.

As a consequence of the so proven practical viability and effectiveness, the proposed control is the subject of a recently filed Patent Cooperation Treaty application.

Future work will be directed at fully automating the calibration of the event-based controller, at completing the performance analysis, and at a more extensive benchmarking. Moreover, the controller will be ported to different processor architectures, with a particular focus on those offering per-core DVFS, and different actuation schemes will be studied, for example addressing the integration of the proposed control

with clock gating, duty cycle modulation, and even the voltage controller, so as to widen the variety of applications for the methodological ideas here proposed.

Also, research may consider taking information from the proposed controller for the use of a task allocator, and in heavy-duty NoCs, one could maybe even treat the communication unities the same as cores, if proper resynchronisers exist to allow for their operating frequency to be modified.

More in general, finally, hardware/software partitioning will be studied to address other computing systems control and design problems, as is definitely seems a promising approach.

REFERENCES

[1] H. Esmaeilzadeh, S. Blackburn, X. Yang, and K. S. McKinley, "Power and performance of native and java benchmarks on 130 nm to 32 nm process technologies," in *Proc. 6th Annu. Workshop Modeling, Benchmarking Simulation*, Saint Malo, France, 2010, pp. 1–10.

[2] M. B. Taylor, "A landscape of the new dark silicon design regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, Sep. 2013.

[3] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annu. Int. Symp. Comput. Archit.*, San Jose, CA, USA, 2011, pp. 365–376.

[4] A. Raghavan *et al.*, "Computational sprinting," in *Proc. 18th IEEE Int. Symp. High Perform. Comput. Archit.*, New Orleans, LA, USA, Feb. 2012, pp. 1–12.

[5] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, "Power management architecture of the 2nd generation intel core microarchitecture, formerly codenamed sandy bridge," in *Proc. 23rd Hot Chips Symp.*, Stanford, CA, USA, 2011, pp. 1–33.

[6] A. Leva, F. Terraneo, and W. Fornaciari, "Event-based control as an enabler for high power density processors," in *Proc. 2nd Int. Conf. Event-Based Control, Commun., Signal Process.*, Krakow, Poland, Jun. 2016, pp. 1–8.

[7] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *Proc. 7th Int. Symp. High Perform. Comput. Archit.*, Nuevo Leone, Mexico, Jan. 2001, pp. 171–182.

[8] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *Proc. 33rd Int. Symp. Comput. Archit.*, Boston, MA, USA, 2006, pp. 78–88.

[9] F. Zanini, D. Atienza, and G. De Micheli, "A control theory approach for thermal balancing of MPSoC," in *Proc. Asia South Pacific Design Autom. Conf.*, Yokohama, Japan, 2009, pp. 37–42.

[10] S. Durand and N. Marchand, "Fast predictive control of micro controller's energy-performance tradeoff," in *Proc. 18th IEEE Int. Conf. Control Appl.*, Saint Petersburg, Russia, Jul. 2009, pp. 314–319.

[11] F. Zanini, C. N. Jones, D. Atienza, and G. De Micheli, "Multicore thermal management using approximate explicit model predictive control," in *Proc. IEEE Int. Symp. Circuits Syst.*, Paris, France, May 2010, pp. 3321–3324.

[12] A. Tilli, E. Garone, M. Cacciari, and A. Bartolini, "Thermal models characterization for reliable temperature capping and performance optimization in multiprocessor systems on chip," in *Proc. Amer. Control Conf.*, Montreal, QC, Canada, Jun. 2012, pp. 4721–4726.

[13] P. Bogdan, R. Marculescu, and S. Jain, "Dynamic power management for multidomain system-on-chip platforms: An optimal control approach," *ACM Trans. Design Autom. Electron. Syst.*, vol. 18, no. 4, 2013, Art. no. 46.

[14] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini, "Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 170–183, Jan. 2013.

[15] S. Murali *et al.*, "Temperature control of high-performance multi-core platforms using convex optimization," in *Proc. IEEE Design, Autom. Test Eur. (DATE)*, Munich, Germany, Mar. 2008, pp. 110–115.

[16] T. Ebi, H. Amrouch, and J. Henkel, "COOL: Control-based optimization of load-balancing for thermal behavior," in *Proc. 8th ACM Int. Conf. Hardw./Softw.-Codesign Syst. Synth.*, Pittsburgh, PA, USA, 2012, pp. 255–264.

[17] F. Mulas *et al.*, "Thermal balancing policy for streaming computing on multiprocessor architectures," in *Proc. IEEE Design, Autom. Test Eur. (DATE)*, Munich, Germany, Mar. 2008, pp. 734–739.

[18] D. Cuesta, J. Ayala, J. Hidalgo, D. Atienza, A. Acquaviva, and E. Macii, "Adaptive task migration policies for thermal control in MPSoCs," in *Proc. VLSI Annu. Symp.*, vol. 105. 2011, pp. 83–115.

[19] J. Zhou, J. Yan, J. Chen, and T. Wei, "Peak temperature minimization via task allocation and splitting for heterogeneous MPSoC real-time systems," *J. Signal Process. Syst.*, vol. 84, no. 1, pp. 111–121, Jul. 2016.

[20] D. Hardy and I. Puaut, "Estimation of cache related migration delays for multi-core processors with shared instruction caches," in *Proc. 17th Int. Conf. Real-Time Netw. Syst.*, Paris, France, 2009, pp. 45–54.

[21] W. Hu *et al.*, "An efficient task mapping algorithm with power-aware optimization for network on chip," *J. Syst. Archit.*, vol. 70, pp. 48–58, Oct. 2016.

[22] W. L. Hung, G. M. Link, Y. Xie, N. Vijaykrishnan, and M. J. Irwin, "Interconnect and thermal-aware floorplanning for 3D microprocessors," in *Proc. 7th Int. Symp. Quality Electron. Design*, San Jose, CA, USA, 2006, pp. 98–104.

[23] S. Niknam, A. Asad, M. Fathy, and A.-M. Rahmani, "Energy efficient 3D hybrid processor-memory architecture for the dark silicon age," in *Proc. 10th Int. Symp. Reconfigurable Commun.-Centric Syst.-Chip*, Tallinn, Estonia, Jun. 2015, pp. 1–8.

[24] F. Tavakkoli, S. Ebrahimi, S. Wang, and K. Vafai, "Analysis of critical thermal issues in 3D integrated circuits," *Int. J. Heat Mass Transf.*, vol. 97, pp. 337–352, Jun. 2016.

[25] R. Redelmeier. *The cpuburn CPU Stress Tester*. [Online]. Available: http://manpages.ubuntu.com/manpages/precise/man1/cpuburn.1.html

[26] D. Brodowski and N. N. Golde. *Linux cpufreq Governors*. Linux Kernel. [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[27] P. Bogdan, "Mathematical modeling and control of multifractal workloads for data-center-on-a-chip optimization," in *Proc. 9th Int. Symp. Netw.-Chip*, Vancouver, BC, Canada, 2015, pp. 21–28.

[28] H. Esen, T. Tashiro, D. Bernardini, and A. Bemporad, "Cabin heat thermal management in hybrid vehicles using model predictive control," in *Proc. 22nd Medit. Conf. Control Autom.*, Palermo, Italy, Jun. 2014, pp. 49–54.

[29] M. Wang and X. Liu, "Robust constrained state feedback IH-MPC based on LMI for a pwr nuclear power plant," in *Proc. 11th World Congr. Intell. Control Autom.*, Shenyang, China, Jun. 2014, pp. 1219–1224.

[30] R. N. Shorten and K. S. Narendra, "Necessary and sufficient conditions for the existence of a common quadratic Lyapunov function for a finite number of stable second order linear time-invariant systems," *Int. J. Adaptive Control Signal Process.*, vol. 16, no. 10, pp. 709–728, Dec. 2002.

[31] R. David, P. Bogdan, and R. Marculescu, "Dynamic power management for multicores: Case study using the Intel SCC," in *Proc. 20th Int. Conf. VLSI Syst.-Chip*, Santa Cruz, CA, USA, 2012, pp. 147–152.

[32] M. Otoom, P. Trancoso, H. Almasaeid, and M. Alzubaidi, "Scalable and dynamic global power management for multicore chips," in *Proc. 6th Workshop Parallel Programm. Run-Time Manage. Techn. Many-Core Archit.*, Amsterdam, The Netherlands, 2015, pp. 25–30.

[33] A. Leva and A. V. Papadopoulos, "Tuning of event-based industrial controllers with simple stability guarantees," *J. Process Control*, vol. 23, no. 9, pp. 1251–1260, Oct. 2013.

[34] M. Akar and K. S. Narendra, "On the existence of a common quadratic Lyapunov function for two stable second order LTI discrete-time systems," in *Proc. Amer. Control Conf.*, Arlington, VA, USA, Jun. 2001, pp. 2572–2577.

[35] *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures*. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf

[36] M. D. Marieska, P. G. Hariyanto, M. F. Fauzan, A. I. Kistijantoro, and A. Manaf, "On performance of kernel based and embedded real-time operating system: Benchmarking and analysis," in *Proc. Int. Conf. Adv. Comput. Sci. Inf. Syst.*, Jakarta, Indonesia, Dec. 2011, pp. 401–406.

[37] Intel Corporation. *Linux Thermal Daemon*. [Online]. Available: https://01.org/linux-thermal-daemon

[38] Intel Corporation. *Intel® Math Kernel Library Benchmarks*, accessed on Mar. 3, 2017. [Online]. Available: https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download

[39] J. Henkel *et al.*, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends," in *Proc. 50th Design Autom. Conf.*, Austin, TX, USA, 2013, p. 99.

[40] R. Keyes, "Physical limits of silicon transistors and circuits," *Rep. Prog. Phys.*, vol. 68, no. 12, pp. 2701–2746, 2005.

Alberto Leva received the Laurea degree (*summa cum laude*) in electronic engineering from the Politecnico di Milano, Milan, Italy, in 1989.

In 1991, he joined the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, where he is currently an Associate Professor of Automatic Control. His current research interest includes methods and tools for the automatic tuning of industrial controllers and control structures, process modeling, simulation and control, particularly within the object-oriented paradigm, object-oriented modeling, and control of energy systems with particular reference to large grids, and advanced tolls and methods for control education, control and control-based design of computing systems, addressing in a system- and control-theoretical manner problems, such as scheduling, resource allocation, time synchronisation in wireless sensor networks, thermal and power/performance management, performance-driven software adaptation, and service composition.

Federico Terraneo received the B.Sc. and M.Sc. degrees in computer engineering and the Ph.D. degree in information technology from the Politecnico di Milano, Milan, Italy, in 2015.

He currently holds a post-doctoral position with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano. His current research interests include embedded systems and the application of principles of control theory to the design of software systems. Since 2008, he has been the main developer and maintainer of the Miosix embedded operating system.

Irene Giacomello received the B.Sc. and M.Sc. degrees in automation engineering from the Politecnico di Milano, Milan, Italy, in 2012 and 2016, respectively. Her M.Sc. thesis was focused on thermal management for microprocessor.

She was with Hilscher Italia srl, Vimodrone, Italy, as a Sales Engineer. She is currently an Instrumentation and Control Engineer with Simeco SpA, Milan. Her current research interest include engineering project in the oil and gas and the chemical/pharmaceutical domains, at the national and international level.

William Fornaciari received the M.S. and Ph.D. degrees from the Politecnico di Milano, Milan, Italy.

He is currently an Associate Professor with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy. He has authored six books and over 200 papers and holds three patents on low-power solutions. He cooperated for 20 years with the Technology Transfer Center of POLIMI, actively cooperating with companies to the development of leading edge products. He has been involved in 18 EU funded projects. His research interests cover multi-many cores, network on chip, low power design, run time resource management, wireless sensor networks, embedded systems, and thermal management. He served as the Project Technical Manager of 2PARMA (ranked as success story by the EU) and he coordinated the HARPA Project, where he filed a PCT patent on thermal management.

Dr. Fornaciari received the 2016 HiPEAC Technology Transfer Award for the output of the FP7 CONTREX Project, five Best Paper Awards, and one certification of appreciation from the IEEE.