Chapter 5
Event-Based Thermal Control for High Power Density Microprocessors

Federico Terraneo, Alberto Leva, and William Fornaciari

5.1 Introduction

The semiconductor industry is currently facing significant roadblocks to deliver improvements in microprocessor performance. One of these is efficient dissipation of the produced power. The failure of Dennard scaling [2] in deep nanometre architectures has resulted in an ever-worsening power density increase, but heat dissipation strategies have not kept the pace. This fact has led to the well-known dark silicon problem [3, 9], where power and thermal constraints limit the number of transistors that can be operated to an ever-decreasing fraction.

Research to overcome this limitation is divided in two complementary directions. One aims at increasing the power dissipation capability of integrated circuits through improved thermal design and heatsinking; the other one addresses the development of microarchitectural improvements and run-time policies that can increase the power efficiency.

In such a scenario, thermal management is a fundamental design challenge that plays a key role in counteracting the variability encountered in current hardware and software architectures. Thus, effective dynamic (or, in other words, run-time) thermal management solutions can push a many-core platform to its maximum performance subject to the constraint imposed by the need to remain within safe operating temperatures.

As part of the HARPA project, thermal management has been addressed both from a high-level perspective, through the TEMPURA policy at the HARPA-OS level, and at the firmware level of HARPA-RT, with the event-based thermal controller presented in this chapter. Thermal management at the HARPA-OS level is

F. Terraneo (✉) · A. Leva · W. Fornaciari
POLIMI, Milan, Italy
e-mail: federico.terraneo@polimi.it; alberto.leva@polimi.it; william.fornaciari@polimi.it

© Springer International Publishing AG, part of Springer Nature 2019
W. Fornaciari, D. Soudris (eds.), *Harnessing Performance Variability in Embedded and High-performance Many/Multi-core Platforms*,
https://doi.org/10.1007/978-3-319-91962-1_5

targeted at use cases where the platform is thermally intrinsically safe, for example because it is already endowed with a hardware thermal protection, and the only goal is to enhance reliability through an added coarse-grained management layer. Thermal management at the HARPA-RT level, conversely, is dedicated to perform fine-grained thermal control at the timescale required by the existing hardware and software variability.

5.2 A Brief Overview of the Thermal Issue in Microprocessors

Microprocessors and in general CMOS integrated circuits consume electrical power through several mechanisms.

First and foremost, switching power is dissipated when individual transistors close causing a current flow to charge the gate capacitance of the transistors to which they are connected. This power contribution depends on four main factors which are related by the well-known formula:

$$P = \frac{1}{2}\alpha C V_{dd}^2 f. \tag{5.1}$$

Of these four terms, the load capacitances C are subject to manufacturing process variability, but most importantly the switching activity α is subject to large run-time variability due to the workload experienced by the microprocessor. For example, during a cache miss the functional units of a core may be mostly idle, thus causing an abrupt drop in the current consumption. The other two terms, voltage and frequency, provide instead a knob, the well-known DVFS through which it is possible to control the system behaviour.

Leakage power is instead caused by deep nanometre transistors deviating from the behaviour of ideal switches, and causing a significant current flow even in the open state. Leakage power depends on the physical conditions of the transistors, which include process variability and, remarkably, temperature. Thus, although leakage power does not directly depend on the workload being executed by the microprocessor, it depends indirectly from it through the temperature rise caused by the switching power.

Other mechanisms that cause power dissipation include resistive losses in the on-chip metal interconnection, which again depend on the current flowing due to the other dissipation mechanisms.

One important fact to notice is that the power consumption of microprocessors is not constant. Active power causes current consumption variations at the timescale of the individual clock edges and the switching activity is heavily influenced by factors including microarchitectural variability induced by cache misses, code patterns exercising the functional units in different ways (think floating point intensive vs.

control flow intensive code fragments), inter-core communication and, at a higher level, OS task scheduling and time-varying workload.

All these factors add up and contribute to induce significant power variations ranging from the nanoseconds to the seconds timescale. Although smoothed by the decoupling capacitors always present in a microprocessor design, power dissipation has significant frequency content in a very broad range, and as we will see, this impacts the design of thermal control policies.

5.2.1 The Thermal Dissipation Problem in a Nutshell

The electrical power consumed by integrated circuits is converted into thermal power, and has to be efficiently dissipated. From a physical perspective, most of the power consumed by an integrated circuit is dissipated in the active silicon layer, whose height is only a fraction of the total chip thickness, and the chip thickness is already a small dimension, usually in the range of a few hundred micrometres. This localised heating causes the two main challenges of thermal control: high power density and fast thermal dynamics.

The first issue is mainly of concern to the research lines trying to design better thermal dissipation solutions, while the second is what shapes the requirements of thermal control policies.

Using the well-known electrical equivalent of thermal circuits, it is possible to sketch a simplified thermal model for a generic microprocessor (Fig. 5.1). This model is too simplistic for simulation, but its abstract nature is well suited to aid the reader in gaining a quick understanding of the involved phenomena.

The power generated by the processor (here assumed spatially uniform for simplicity) is modelled as a current generator P_{gen}. This power is first *quickly* absorbed by the thermal capacitance of the silicon chip C_{silicon}, and dissipated through the thermal resistance from the chip to the heatsink R_{silicon}. This in turn *slowly* heats up the heatsink thermal capacitance C_{sink}, which finally exchanges heat convectively with the ambient.

The key aspect that can be gained from this model is in the two highlighted adjectives of the previous sentence. The thermal dissipation stack can be seen, as a first approximation, as a two-stage RC filter over the microprocessor power consumption. Since the microprocessor chip has such a small volume, its thermal

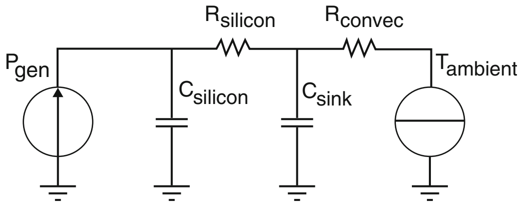


Fig. 5.1 A simplistic model of the heat dissipation stack for a microprocessor

capacitance is quite low, and thus the chip temperature can change quite rapidly. The heatsink, on the other hand, is a large block of metal and its temperature changes much more slowly.

As we have seen before, the power of a microprocessor exhibits significant variations in a wide range of timescales, thus we are sure that, in real-world workloads, both thermal dynamics will be exercised, and since the objective of thermal control is that of controlling the chip temperature, not the heatsink one, control policies need to act at a faster timescale than the fastest involved dynamic to be of any effectiveness.

But, how fast can on-chip temperature vary? The best way to answer this question is through an experiment. To model a dynamic system, in the sense given to this term by control theory, it is possible to feed it with some reference signal and measure the output. Of the many possibilities, the step and impulse response are two of the most suitable ones. Their main advantage is the easy interpretation of the result as well as the possibility to reproduce the behaviour of the system when subject to arbitrarily complex signals by suitable convolutions. Thus, although real-world CPU workloads are far more complex, we have chosen a step response for the presented experiment.

The experiment was performed on an Intel Core-i5 6600K running Linux, and to produce the step in power consumption the program cpuburn [8] was used. The result is shown in Fig. 5.2.

The cpuburn program was started 5 s after temperature logging was enabled. In addition, it was briefly stopped for 500 ms at $t = 23$ s and $t = 41.5$ s to show an important point that will be discussed shortly.

Due to the small thermal capacitance of the silicon C_{silicon} and the non-negligible thermal resistance towards the heatsink R_{silicon}, the processor temperature rises from 54 to 70 °C in 50 ms, a 16 °C increase, and reaches 80 °C after only 600 ms since cpuburn was started. After that, the heat produced by the processor starts heating up the heatsink, which in turn drags the chip temperature even higher, although more slowly. Since this second part of the thermal transient is driven by the heatsink thermal dynamics, it takes roughly 55 s, two orders of magnitude more time for the

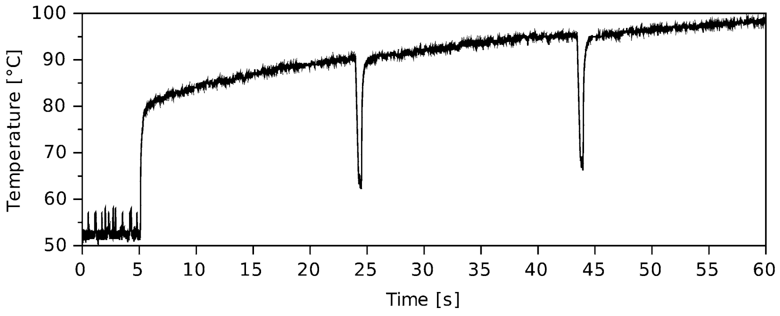


Fig. 5.2 Step response of the heat dissipation stack for an Intel processor

chip temperature to rise from 80 to 99 °C, when the experiment was stopped not to risk damaging the processor.

Summarising, this experiment evidences the existence of two well separate thermal dynamics. When the power is increased stepwise, the temperature will increase at a fast rate at first, and then continue increasing more slowly.

There is just one final remark to be made before considering the effect of these considerations on the design of thermal policies. By observing the behaviour of the system when cpuburn is first started, it can be noticed that although temperature rises at a fast rate, it does so when temperature is low, and when temperature is high, which is where thermal control policies should act, temperature increase is much slower. This is a common objection to the need for fast thermal control policies. Consider a hypothetical policy trying to keep the processor temperature below 85 °C. Since cpuburn is started at $t = 5$ s but the temperature first reaches 85 °C at $t = 10$ s, it appears that the policy has 5 s to react.

This is not true, as the fast temperature rise occurs at low temperatures *only when the heatsink is cold*. Consider again Fig. 5.2 and concentrate on what happens at $t = 23.5$ s. Here, cpuburn has been stopped, but since the heatsink takes a significant time also to cool down, when the power consumed by the processor is increased again, the core temperature reaches 85 °C in just 110 ms, and when the same operation is performed at $t = 42$ s when the heatsink temperature is even higher, only 54 ms are needed to reach 85 °C.

This clearly shows that thermal policies, to be effective, have to respond at most in a few tens of milliseconds.

5.2.2 Thermal Control Policies

We hope that by now we have convinced the reader that one of the most challenging tasks in designing a thermal control policy is that the policy sensing, decision and actuation loop needs to be run at the millisecond timescale due to the need to control the microprocessors' fast thermal dynamics.

A policy can be implemented entirely in hardware, with a dedicated data path and state machine implementing the policy algorithm, or can be implemented in software, where only the temperature sensor and DVFS actuator are hardware components. The main advantage of a hardware implementation is that it frees the microprocessor from the overhead of executing the thermal control algorithm, which given that it needs to be executed every few milliseconds can reduce the performance of applications noticeably. However, the main drawback of a hardware implementation is its inflexibility, as it prevents fine tuning that can be needed to overcome the software and hardware variability.

The proposed event-based thermal controller takes a mixed hardware–software approach to achieve the benefits of both a hardware and a software implementation. An ordinary PID control algorithm has to be executed at a fixed rate, even when temperature is not changing. In the HARPA project, we have explored the powerful framework of event-based control theory, to design a small hardware state machine which generates events only when temperature is changing. These events cause interrupts which in turn execute the software control algorithm. By doing so, the flexibility of a software implementation is preserved, while significantly reducing its overhead.

5.3 A Modelica Thermal Simulation Library

To perform the simulation studies required by the presented research, we used Modelica, which is an object-oriented modelling language.

Two are the main reasons for the choice of object-oriented modelling, and in particular the Modelica language: multi-physics modelling, and the possibility of co-simulating equation-based and algorithmic components. Multi-physics is important for the addressed domain, since achieving a sound validation of a thermal management policy requires to account for the many and diverse installation conditions that a microprocessor can encounter. It is thus necessary to model the chip, the heat sink, possibly the casing and the cooling system, be this just an air mover or a liquid cooler or anything else, and sometimes even the power supply electronics. The co-simulation of components described through equations and through algorithms allows to model the processor and its heat dissipation stack as well as the controller in their most natural representation, and also for the generation of *stimuli* to replicate the behaviour of the various computational loads to which the microprocessor can be subjected. Quite intuitively, a modelling paradigm that allows to do all of the above with a single tool is of great help for the presented research.

The thermal simulation and control has been implemented as a Modelica library, which is composed of three sections briefly described below:

1. *Components.* This section contains the building blocks used by the rest of the library, such as a solid control volume with heat capacity and boundary conductances, well-assessed empirical correlations for convective heat exchange and domain-specific elements like a cooling fan. The same section contains aggregate components, including a three-dimensional array of solid volumes, which is typically used to simulate the temperature distribution into a chip or a heat spreader. One-dimensional elements are also available, being suited to model elements such as heat pipes. The components of this section are directly modelled in the continuous time with differential equations. The separation of the modelling equations from the differential equation solver is in fact another important advantage of adopting an object-oriented paradigm, as this allows to take profit of newly developed solvers with no effort on the part of the analyst.
2. *Controllers.* The main element of this section is the event-based controller presented in Sect. 5.5; the section also contains the models of the aggregate blocks used to compose the devised control architectures, so as to obtain the results of Sect. 5.6. For the reasons explained above, all these models are

algorithmic. In addition, this section includes continuous-time controllers, that are functionally equivalent to the event-based ones just mentioned, and can be used as a baseline for comparison in simulations targeted to control quality assessment.

3. *Stimuli sources.* This section includes ad hoc power signal generators, useful to produce realistic profiles based on conveniently specified characteristics—such as the use over time of arithmetic units, pipelines, cache and so forth—of the possible applications running on the simulated microprocessor.

In developing the library, care was taken to design physical connectors (electric pins, heat ports and so on) compatible with the Modelica Standard Library, and with other libraries that can be usefully coupled to this in a view to widening the set of simulated physics—for example, the ThermoPower library [1] to represent thermal and hydraulic phenomena in cooling systems, including both those comprised in the typical CPU rack as well as for large-scale simulations of data centre air conditioners.

To briefly show the library in action, Fig. 5.3 shows the Modelica scheme of the model of a microprocessor, including the heat spreader which is part of the package, connected to a heat sink, exchanging with air at a prescribed ambient temperature, subjected to a time-varying (in this case, multi-harmonic) profile of generated powers in the absence of any thermal management.

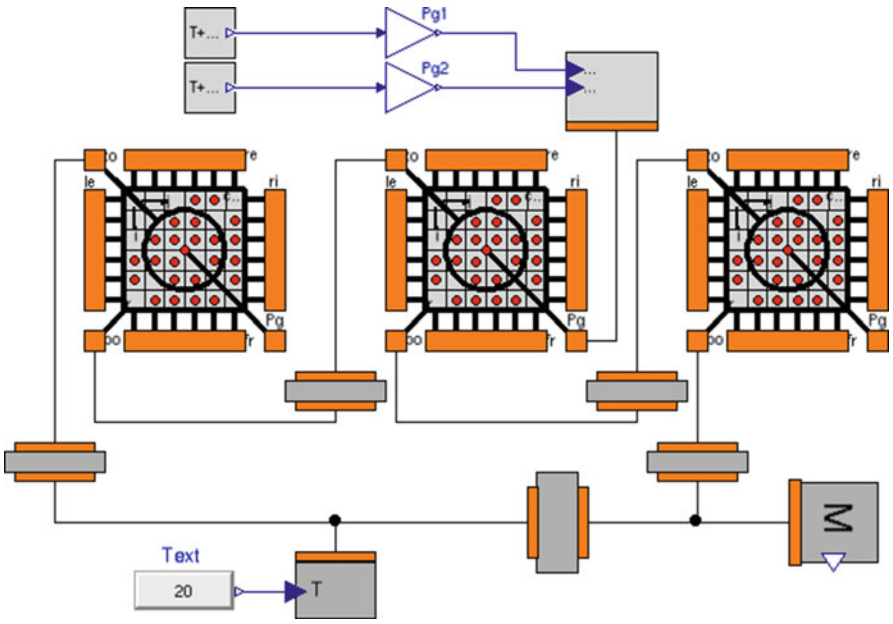


Fig. 5.3 Chip subjected to exogenous powers, open loop—Modelica scheme

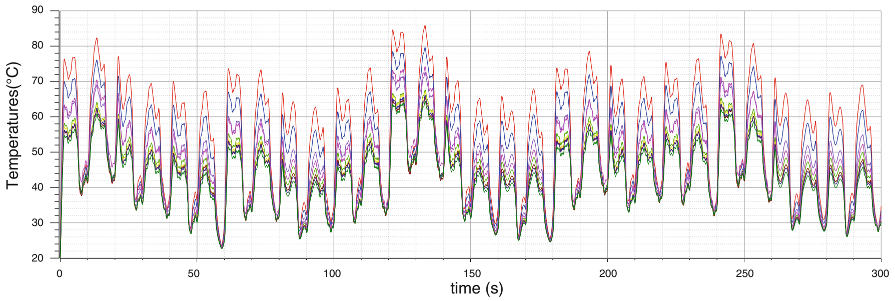


Fig. 5.4 Chip subjected to exogenous powers, open loop—ten simulated temperatures from the grid

Figure 5.4 shows an example of the obtained results, showing the behaviour of selected temperatures in the chip for clarity. Note the large and fast swings, furthermore testifying the criticality of the control problem and the usefulness of the event-based approach.

The simulation of Fig. 5.4, that was done with a 10 × 10 grid and three layers to represent the silicon, package and heat sink, took 23.5 s on a 64-bit i5-based machine. This is about 25 times faster than real time, allowing for extensive test campaigns with an acceptable effort, as well as giving the possibility to test corner cases in the silicon power consumption difficult to exercise using benchmarks on a real machine. As for the spatial discretisation, based on experience we can say that the adopted one is adequate for most control-oriented—i.e. system level—studies. At present, research is nonetheless ongoing towards the use of sparse solvers, in order to allow for significantly more fine-grained simulations, should this be necessary, e.g. for the final validation of a strategy. We would like to stress once again, however, that the tool is already well suitable for control-centred studies, as testified by the experimental results of Sect. 5.6, where the used control system has been assessed by means of the library just described.

5.4 Event-Based Thermal: The Hardware Event Generator

In [7], it is shown and motivated that a decentralised control architecture, devoting one event-based PI controller to each core, can provide efficient enough temperature control. In this and the following section, we describe how the preliminary results of that paper were turned into a fully functional system working on real hardware.

As anticipated in Sect. 5.2.2, hardware–software partition is the proposed solution to achieve the fast response required by the thermal control problem at an acceptable overhead. Specifically, and given the decentralised nature of the overall scheme, each core is equipped with a hardware event generator, to interact with the thermal sensors, and a software controller.

This section deals with the hardware event generator, the following with the software control policy.

The purpose of the hardware event generator is to generate interrupts to run the software control policy only when needed. The design combines a send-on-delta and a timeout policy as follows. The temperature sensor is sampled at a fixed interval q_s, typically in the range from 500 µs to 10 ms. A temperature threshold Δ and a timeout are selected. Every q_s, a new temperature value is read, and if it differs in magnitude from the value when the controller was last run by more than Δ, a threshold event is generated and the controller is run again. If instead the timeout expires, a timeout event is generated and the controller is run. The software controller is informed whether the event is due to a threshold or timeout through a flag bit in a register exposed by the event generator.

The timeout value is dynamically adapted (in software) to satisfy the opposing constraints of control quality and low overhead. The decision to increase or decrease the timeout depends on the reason why the controller was called. In the case of a timeout event, the timeout is increased exponentially, up to a maximum value that in the proposed implementation is 0.5 s. If instead the controller was called due to a threshold event, the timeout is immediately reduced down to the minimum value q_s, forcing the controller to be run again when the next temperature sample is available.

The event generation policy is expected to be implemented in hardware, as a simple state machine connected to a data path to compare the absolute value of the current temperature reading with the one when the controller was last run, hence deciding if an interrupt has to be generated. The timeout can be easily implemented using a hardware counter incremented at a frequency equal to $1/q_s$. Figure 5.5 shows a high-level diagram of the proposed event generator, detailing the registers used for the required communication with the software interrupt routine.

To have an estimate of the area and power consumption of the proposed hardware event generator, we implemented and simulated it in RTL Verilog, and then synthesised it in Cadence Encounter using the NAND Gate Liberty standard cell library. Synthesis results were obtained considering an operating voltage of 1.1 V and a clock frequency of 667 MHz, which yielded a per-core area and power

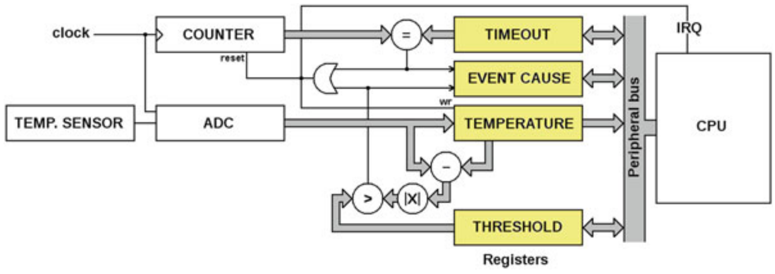


Fig. 5.5 High-level logic scheme of the hardware event generator

overheads of 159 μm^2, 471 μW, respectively. The proposed solution has thus a negligible impact compared to the area and power consumption of modern CPU cores.

5.5 Event-Based Thermal: The Software Control Policy

In this section we describe the software part of the proposed solution, that is, the configurable control policy.

5.5.1 Control Structure and Closed-Loop System

As discussed in [7], control events are generated by the temperature sensor with a send-on-delta policy, that now needs to be described more in detail. It is not the purpose of this section to describe the internals of the sensor, suffice in this respect to say that it has an internal "fast" sampling mechanism acting at period q, based on which an event is possibly triggered. We thus introduce an index h to count the mentioned fast sensor samplings.

As anticipated, we adopt a PI control law. In state space representation, this takes the form of two multiply-and-accumulate operations, that is,

$$\begin{cases} x_R(k) = x_R(k-1) + b_R(w(k-1) - y(k-1)) \\ u(k) = x_R(k) + d_R(w(k) - y(k)) \end{cases} \tag{5.2}$$

where w, y, u and x_R are the set point, the controlled variable, the control signal and the PI state variable, while k is the *control* discrete-time index—i.e. it counts the interventions of the controller, not the samplings of the sensor. To keep the notation as light as possible, in the following when we need to locate the k-th event in the fast sampling, we shall indicate its index with $h(k)$, while when this is not needed we shall write for the generic signal $v(k)$ to mean $v(h(k))$.

Based again on [7], we describe the controlled system with a first-order, SISO, strictly proper model. Also, we make the assumption that the fast sampling period q is small enough for a discrete-time model at step q to be practically equivalent to a continuous-time one. This said, we write

$$\begin{cases} x_P(h) = a_P x_P(h-1) + b_P u(h-1) \\ y(h) = c_P x_P(h) \end{cases} \tag{5.3}$$

where x_P is the state of the controlled system. The controller (5.2) is realised in event-based form as follows. At the k-th event, the sensor provides the current and also the previous sample of the controlled variable, i.e. $y(h(k))$ and $y(h(k) - 1)$.

The controller then computes $u(k)$ and keeps it constant till the last event, that is,

$$u(l) = u(h(k-1)) = u(k-1), \quad l = h(k-1)\dots h(k)-1. \tag{5.4}$$

In accordance with the hold operation above, prior to computing $u(k)$, the state x_R is made consistent with $y(h(k)-1)$. Assuming that the set point is modified—or sensed—only at events, which is sensible as it is seldom (if ever) modified, from the controller's viewpoint we have

$$w(l) = w(h(k-1)) = w(k-1), \quad l = h(k-1)\dots h(k)-1. \tag{5.5}$$

Given the above, the controller state at time $h(k)-1$ is

$$\begin{aligned} x_R(h(k)-1) &= u(h(k)-1) \\ &\quad -d_R(w(h(k)-1) - c_P x_P(h(k)-1)) \\ &= u(k-1) - d_R(w(k-1) - c_P x_P(h(k)-1)), \end{aligned} \tag{5.6}$$

and therefore the same state at the k-th event is

$$\begin{aligned} x_R(k) &= x_R(h(k)-1) + b_R(w(k-1) - c_P x_P(h(k)-1)) \\ &= u(k-1) - d_R(w(k-1) - c_P x_P(h(k)-1)) \\ &\quad +b_R(w(k-1) - c_P x_P(h(k)-1)) \\ &= u(k-1) + (b_R - d_R)(w(k-1) - c_P x_P(h(k)-1)). \end{aligned} \tag{5.7}$$

Now, since

$$u(k-1) = x_R(k-1) + d_R(w(k-1) - c_P x_P(k-1)) \tag{5.8}$$

we get

$$\begin{aligned} x_R(k) &= x_R(k-1) + d_R(w(k-1) - c_P x_P(k-1)) \\ &\quad +(b_R - d_R)w(k-1) \\ &\quad -(b_R - d_R)c_P x_P(h(k)-1) \\ &= \dots \\ &= x_R(k-1) + b_R w(k-1) - d_R c_P x_P(k-1) \\ &\quad +(b_R - d_R)c_P x_P(h(k)-1)). \end{aligned} \tag{5.9}$$

If we evidence the variation of x_P from the $(k-1)$-th event till "immediately" (i.e., q) before the k-th as

$$x_P(h(k)-1) = x_P(k-1) + \delta x_P(k-1), \tag{5.10}$$

where the index attributed to δx_P is $k-1$ as that quantity is known before the k-th event, we obtain

$$\begin{aligned} x_R(k) = {} & x_R(k-1) - b_R c_P x_P(k-1) + b_R w(k-1) \\ & +(d_R - b_R) c_P \delta x_P(k-1). \end{aligned} \tag{5.11}$$

Reasoning in an analogous way for the state of the controlled system, we have

$$\begin{aligned} x_P(k) &= a_P x_P(h(k)-1) + b_P u(h(k)-1) \\ & \quad a_P x_P(h(k)-1) + b_P u(k-1) \\ & \quad a_P (x_P(k-1) + \delta x_P(k-1)) + b_P u(k-1) \\ &= \dots \\ &= (a_P - b_P d_R c_P) x_P(k-1) + b_P x_R(k-1) \\ & \quad + b_R w(k-1) + (d_R - b_R) c_P \delta x_P(k-1). \end{aligned} \tag{5.12}$$

Thanks to the holding mechanism described above, and to the pre-update of the x_R in accordance with the (additional) past value of the controlled variable transmitted by the sensor, we can represent the closed-loop system in the k index—i.e. counting the events independently of their distance in the constant-rate sampling at step q—by writing

$$\begin{cases} x(k) = Ax(k-1) + bw(k-1) + f \delta x_P(k-1) \\ o(k) = Cx(k) + dw(k) \end{cases} \tag{5.13}$$

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

$$\begin{aligned} & A = \begin{bmatrix} a_P - b_P d_R c_P & b_P \\ -b_R c_P & 1 \end{bmatrix}, \quad b = \begin{bmatrix} b_P d_R \\ b_R \end{bmatrix}, \\ & f = \begin{bmatrix} a_P \\ (d_R - b_R) c_P \end{bmatrix} \quad C = \begin{bmatrix} c_P & 0 \\ -d_R c_P & 1 \end{bmatrix}, \quad d = \begin{bmatrix} 0 \\ d_R \end{bmatrix} \end{aligned} \tag{5.14}$$

Finally, and again in accordance with [7], we endow the event triggering mechanism with a timeout, i.e. a time span after which the sensor performs a new transmission unconditionally. This is useful as a keep-alive measure, and in no sense impairs our results.

5.5.2 Tuning, Stability, Robustness and Performance

The synthesis of (5.2) is done in the discrete-time domain. Doing so, a natural way to tune the PI controller is to prescribe the eigenvalues of matrix A in (5.14). For example, setting

$$b_R = \frac{1 + e_1^2 - 2e_1 - e_2}{b_{PCP}}, \quad d_R = \frac{1 - 2e_1 + a_P}{b_{PCP}} \tag{5.15}$$

makes those eigenvalues $e_1 \mp \sqrt{e_2}$. It is therefore straightforward to make the closed-loop system asymptotically stable, while having the integral action in the controller structurally guarantee zero steady-state error, which also implies complete asymptotic rejection of constant disturbances.

Given the simplicity of the controlled system model described above, evaluating its stability degree, hence its robustness, is straightforward. More difficult is an assessment of its performance, however: it is easy to compute how many k steps—i.e. events—the free motion of (5.13) needs to converge to zero within a given tolerance, but this gives no information about the *time* duration of that transient, since (5.13) disregards for the interval between two adjacent events. This aspect deserves a few more comments.

First, denoting by $\overline{N}_c$ the number of steps taken by the free motion of (5.13) to converge to zero, a worst-case estimation of the closed-loop settling time (an adequate performance indicator for our purposes) is readily obtained as $\overline{N}_c$ multiplied by the prescribed timeout. This is inherently very pessimistic, however, because if most of the events during a transient are timeouts, then either the PI is poorly tuned or the send-on-delta threshold is not adequate.

An inherently optimistic settling time estimate, conversely, is obtained by supposing that the number of h steps between every two subsequent events equals the send-on-delta threshold divided by the one-step variation of y taken at the h step corresponding to the first of the two, of course rounded to the nearest greater integer. The settling time estimate is then the sum of so approximated inter-event periods up to the $\overline{N}_c$-th.

We omit further details on this matter because research is still underway, and in any case the results obtained to date are satisfactory, generally outperforming state-of-the-art alternatives.

In practice, then, to calibrate the control system, one can obtain the required model by subjecting the processor to a load and a DVFS step; from the so-gathered response data, and given the fast sampling time, model (5.3) is parameterised straightforwardly with any of the numerous methods available in the literature.

5.5.3 Implementation and Simulation-Based Assessment

When the proposed solution is implemented in a multi-core or many-core processor, the sensor and hardware event generator are conceived to be implemented in hardware (although for the tests of Sect. 5.6 we had to emulate this). The event-based controller, conversely, is expected to be realised in software. As anticipated, events are triggered when the present temperature differs in magnitude more than Δ from that at the last event, or when a configurable timeout T has elapsed since that event. If an event is caused by a timeout, the software controller increases the value of T up to a prescribed maximum. If on the contrary the event is caused by a temperature variation, T is brought back down to a prescribed minimum. This is substantially the same event triggering mechanism of [7], thus we omit further details if not for noticing that the conjectures made therein are now confirmed on real hardware.

The used knob is the DVFS mechanism present in any modern processor. More precisely, the event-based controller produces as output a frequency value. This value is decided based on the current temperature, the state of the controller and the desired set point, which is set to a value slightly lower than the acceptable maximum temperature. Since the controller produces a frequency value, the setting of the corresponding voltage is left to the installed electronics.

Modern operating systems also have power-performance controllers which measure the system load and use a “governor” to act on DVFS to increase the frequency when the load is high and decrease it in the opposite case, to save energy and useless heating while keeping the system responsive.

The presented thermal management scheme can seamlessly integrate with the power/performance system via an override configuration: the lower frequency is selected and applied between the one requested (based on the load) by the power/performance governor, and the one computed by the temperature PI controller, the state of which is updated accordingly. This configuration naturally makes power/performance prevail when the temperature is well below the limit, and thermal control takes action and rules in the opposite case.

To give an example of the assessment activities carried out by means of the presented Modelica simulation library, we show a test for the event-based controller, completed with all the features just listed, subjected to a suddenly variable computational system load. Figure 5.6 shows the controlled core temperature and the maximum admissible one T_{thr}, while Fig. 5.7 reports the frequency output from the controller, normalised in the (0,1) range, which corresponds to the available frequency span. The set point given to the event-based PI is computed as the maximum admissible temperature minus twice the send-on-delta threshold—an empirical rule of thumb that proved successful in practice.

As can be seen, despite the large and abrupt load variations, the objective of operating at the maximum speed required by the load, while not violating (if not in a practically negligible manner, which is admissible) the temperature limit, is met. In fact, given the typically low resolution of the temperature sensors embedded in microprocessors, one could even use the maximum admissible temperature as set point directly, and the benefits of the proposed control for the device would still be obtained in full.

To weigh the overhead of the proposed control scheme, Fig. 5.8 reports the inter-event times along the test. As can be seen, the fast reaction time of the proposed controller has been obtained by using a fast controller activation interval only when needed, i.e. when rapid temperature changes occur. The inter-event period is instead increased up to 100 ms for a significant part of the simulation. Thus, the remarkable

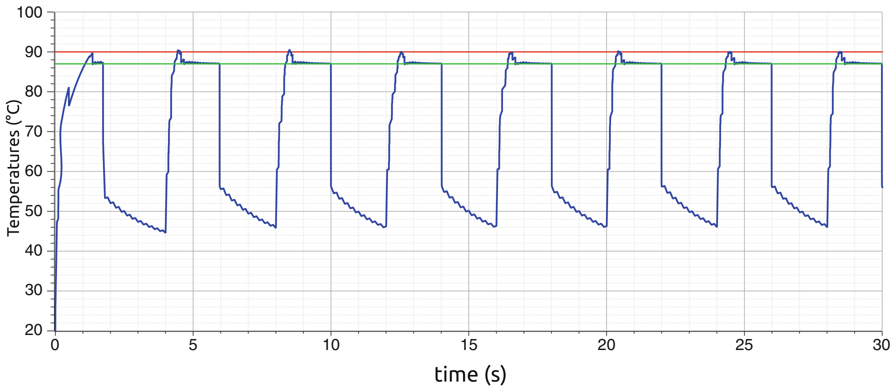


Fig. 5.6 Example of control assessment simulation—processor temperature (blue) versus the maximum admissible one (red) and the set point provided to the event-based controller (green)

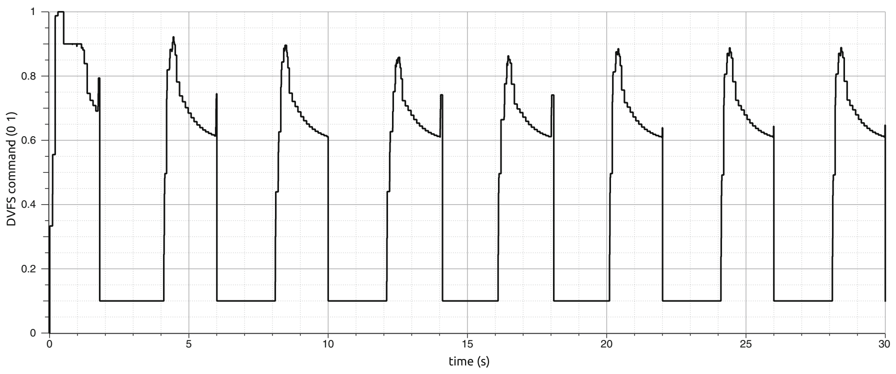


Fig. 5.7 Example of control assessment simulation—DVFS command normalised into the (0, 1) range

control performance of Fig. 5.6 is obtained with just 809 events in the 30 simulated seconds—i.e., about 27 events per second on average. This fact, confirmed in practice as in the performed experiments we could get even lower events/second ratios, indicates that also the objective of a low system overhead is achieved.

5.6 Experimental Results

The system we used for experimental tests is a PC with an ASUS Z170K motherboard equipped with an Intel i5-6600K microprocessor and a Cooler Master finned heat-pipe sink. The fan was removed from the heatsink during the tests, to

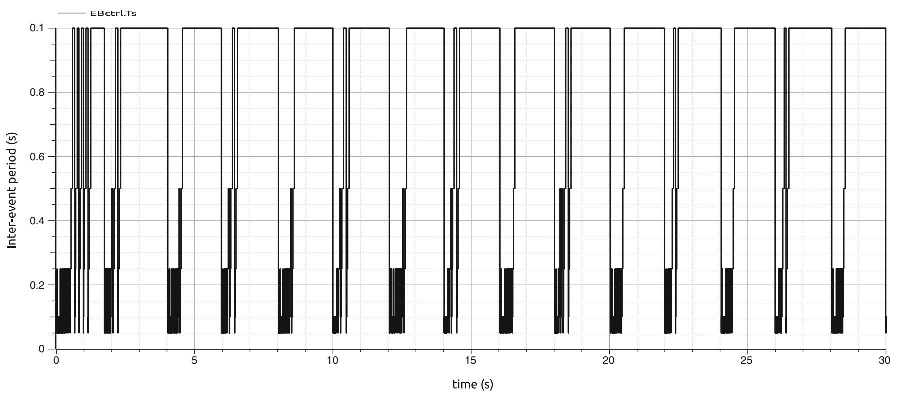
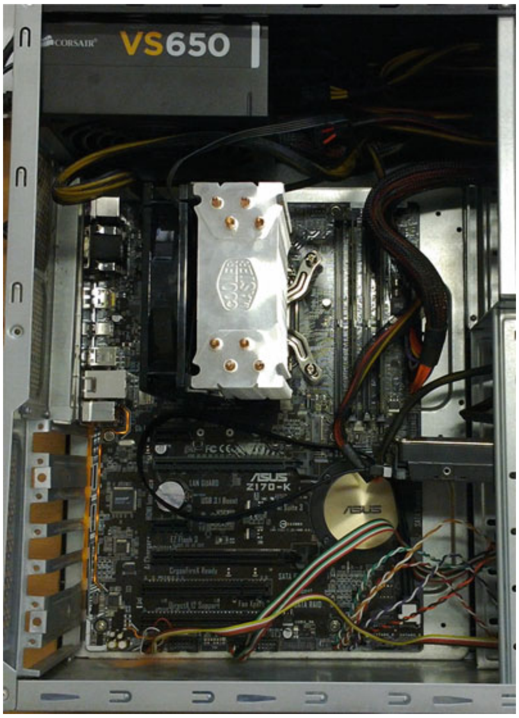


Fig. 5.8 Example of control assessment simulation—inter-event time

Fig. 5.9 Experimental setup



simulate an underdimensioned heatsink as well as to better assess the controller under extreme heat exchange conditions. A photo of the experimental setup is shown in Fig. 5.9.

The experiments were performed using Ubuntu Linux version 15.10. The kernel modules devoted to thermal and power/performance management were removed, and the userspace governor was selected, so as to avoid any operating system interference in the tests. Also, for a meaningful assessment, we disabled the Turbo Boost feature of the microprocessor, as it is based on internal hardware that is not required by our solution, and could unpredictably alter the results. As disabling Turbo Boost limits the maximum CPU frequency, we overclocked it to operate up to 4.2 GHz. The processor has internal thermal protections, named TM1 and TM2, that engage when the critical temperature (in the used processor, 100 °C) is reached. These protections were left in place, as they have no role if not in a thermal emergency, that however with the proposed controller never happened. Under this configuration, the processor DVFS is not controlled by any policy other than the proposed controller, thus allowing to collect repeatable results.

To complement the thermal controller with a power-performance one in order to perform more representative experiments, we implemented a simple power-performance policy setting the DVFS to a filtered value of the load. The actuation values of the thermal and power-performance policies are combined as described in Sect. 5.5.3.

The event-based controller has been implemented as a userspace C++ program, reading the temperature and performing the DVFS actuation via the MSR [6] interface offered by Intel processors. The choice of using the MSR interface allows fast and low-jitter sensing and actuation. The choice of designing the controller in userspace conversely allows to conveniently log the operation of the controller, including the events, temperatures and DVFS actuations, although at the price of a higher overhead. As it is not possible to implement the event generation state machine in hardware on a commercial processor, it has been emulated in software. Every 5 ms, the temperature sensors are read, and a software state machine decides whether or not to call the controller. Finally, though the processor has a per-core temperature sensor, there is only one DVFS actuator for all cores. For this reason, a single loop is used, using the instantaneous maximum among all four temperatures as the controlled one.

5.6.1 Control Quality and Overhead

The first test is aimed at assessing the ability of the proposed controller to keep the temperature at or below the prescribed set point. For this test, the `cpuburn` application [8] is launched on all cores. This application is specifically designed to maximise the processor current consumption. To test the response to fast power transients and provide a more diverse load profile than a constant power, `cpuburn` is periodically stopped and restarted. Of course, such a load is not representative of the normal operation of a processor, but step-like *stimuli* are well suited to compare controllers by transient examination.

In this test, the temperature set point has been set to 90 °C, and Fig. 5.10 reports the results. The top plot shows the controlled temperature, the centre plot the DVFS actuation and the bottom one the CPU load. When the measured temperature is lower than the limit, the power-performance policy prevails, and frequency

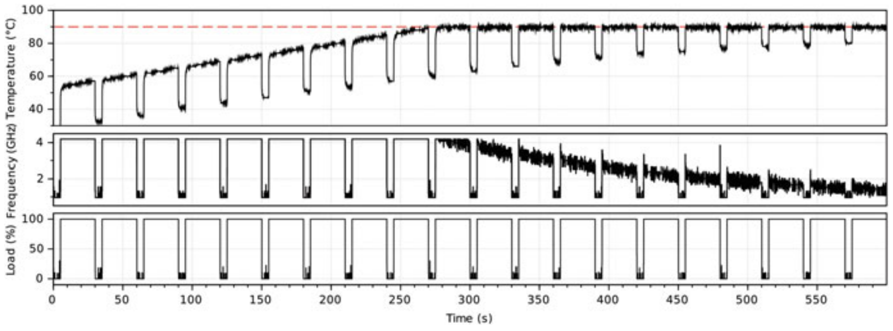


Fig. 5.10 Experimental test 1—controlled temperature and threshold (top), DVFS command (centre) and computational load (bottom)

follows the load pattern, saving power in idle periods. When the `cpuburn` applications are started and stopped, fast temperature transients, up to 30 °C/100 ms, are observed, while the slow rising trend is due to the heat sink warming up. Once the temperature reaches the prescribed set point, the thermal controller starts to reduce the frequency, and its action becomes stronger over time to compensate for the heat sink temperature trend. The controller successfully keeps the temperature at or below the prescribed limit, with an average of only 14 events per second.

5.6.2 Comparison with the State of the Art

The proposed controller has been compared against `thermald` [5], a service for the Linux operating system to control the CPU temperature, recently developed by Intel; `thermald` is configured through a file where it is possible to set the limit temperature. Internally, it uses a PID controller calibrated at start-up not to require control knowledge on the part of the user.

The first benchmark used for comparison is based on `cpuburn` as the previous test. In this case, `cpuburn` is run continuously for 260 s, to obtain a constant CPU power. After that, it is periodically stopped and restarted to introduce power transients.

Figure 5.11 shows the results. Both controllers perform acceptably, although `thermald` causes temperature fluctuations with an amplitude of 10 °C, even when the CPU power is almost constant. The proposed controller limits such fluctuations to $+2/-3$ °C and is limited mainly by the sensor noise. The proposed controller can achieve this level of performance in controlling the temperature at just 22 events/s. When power transients are introduced, both controllers can prevent temperature peaks.

The second benchmark used is taken from the Intel optimised `LINPACK` parallel suite [4]. The chosen benchmark consists in solving a system of linear equations,

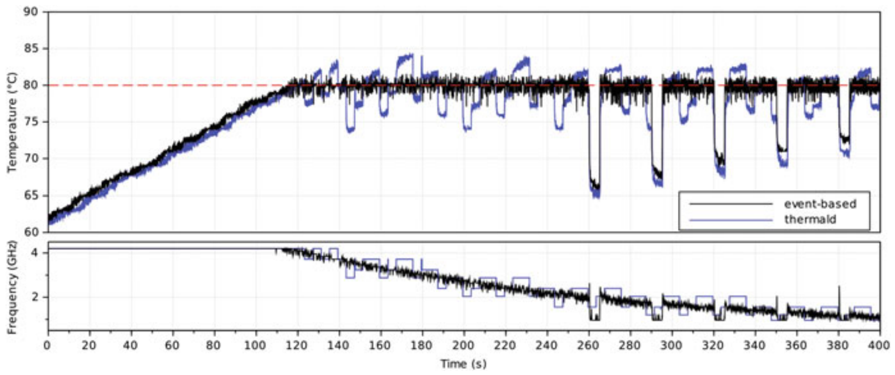


Fig. 5.11 Experimental test 2—comparison between the proposed controller and `thermald` when running `cpuburn`. Controlled temperature and threshold (top) and DVFS command (bottom)

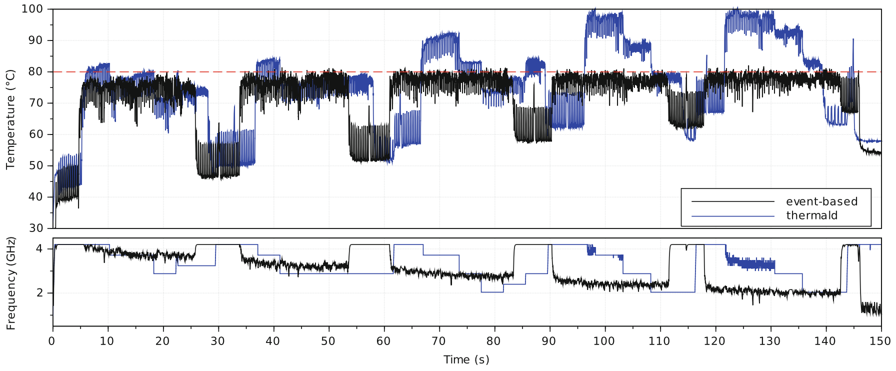


Fig. 5.12 Experimental test 3—comparison between the proposed controller and `thermald` when running `LINPACK`. Controlled temperature and threshold (top) and DVFS command (bottom)

and stresses the CPU floating point pipeline considerably. This test is relevant because `LINPACK` alternates execution phases that cause very different cache miss rates; thus, the CPU is never idle, which means 100% *load* throughout the test, but *power* varies significantly.

As a consequence, see Fig. 5.12, the two controllers yield significantly different results. The event-based one successfully keeps the temperature set point. The only adverse effect of the mentioned power bursts is an increase in the number of thermal events, on average 61 per second. `thermald` is instead simply not capable of keeping the temperature below the limit, due to its slow actuation. It succeeds in the first two iterations, when the heat sink is warming up, but when the slow temperature trend has raised near enough to the limit, it fails to react timely, and the CPU temperature reaches 100 °C between 96 and 98 s from the experiment start, and again between 121 and 131 s. In those time frames, the hardware thermal protection is engaged, as shown by the evident high-frequency content of the DVFS command.

As a final comment, while `cpuburn` executes its task in an endless loop until terminated, the `LINPACK` benchmark performs a finite amount of work, and thus the completion time can be used as a metric of the CPU performance. During the first and second iteration, when both controllers succeed in keeping the temperature limit, the event-based controller results in an increased computational speed, finishing the second iteration after 53.5 s compared to 58 s with `thermald`—a 7.8% improvement. During the following iterations, the execution is slightly faster with `thermald`, but this is because it exceeds the selected temperature limit, and not negligibly.

Summarising, the proposed event-based scheme is capable of controlling temperature at the speed required to counteract fast transients, with all the advantages of software configurability and openness to new control laws, with full power/performance integration, and with a very low overhead.

5.7 Conclusions

In this chapter, we presented the event-based digital controller for the thermal management of microprocessors that has been developed as part of the HARPA project. Peculiar of our proposal is the integration into a unitary scheme of temperature control and power/performance trade-off. We carried out a stability analysis, and set preliminaries for a performance one. The proposed controller was tested and assessed by means of a comprehensive Modelica library, that we created by extending previous works, and can serve as a solid benchmark, open to multi-physics settings, for control studies on the matter addressed herein. We also showed some experimental results obtained on a modern processor architecture and compared our solution to the state of the art, in the form of a recently proposed operating system thermal daemon to be used jointly with classical power/performance governors.

Our proposal behaves significantly better than the available alternatives and is easy to set up, calibrate and maintain. We can thus state that the presented control system acts as a technology enabler by allowing the safe operation of high-density processor, also for what concerns their fastest thermal dynamics. As a final note, concerning the proposed control solution, a PCT (Patent Cooperation Treaty) application has been filed.

References

1. Casella, F., & Leva, A. (2006). Modelling of thermo-hydraulic power generation processes using Modelica. *Mathematical and Computer Modelling of Dynamical Systems, 12*(1), 19–33.
2. Dennard, R., Gaensslen, F., Rideout, V., Bassous, E., & LeBlanc, A. (1974). Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits, 9*(5), 256–268.
3. Esmaeilzadeh, H., Blem, E., Amant, R. S., Sankaralingam, K., & Burger, D. (2012). Dark silicon and the end of multicore scaling. *IEEE Micro, 32*, 122–134.
4. Intel Corporation. Intel optimized LINPACK benchmark. https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download
5. Intel Corporation. Linux thermal daemon. https://01.org/linux-thermal-daemon
6. Intel 64 and IA-32 Architectures Software Developers Manual. http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-system-programming-manual-325384.html
7. Leva, A., Terraneo, F., & Fornaciari, W. (2015). Event-based thermal control for high-density processors. In *Proceedings of 1st International Conference on Event-based Control, Communication, and Signal Processing*, Krakow (pp. 1–8).
8. Redelmeier, R. The `cpuburn` CPU stress tester. http://manpages.ubuntu.com/manpages/precise/man1/cpuburn.1.html
9. Taylor, M. (2013). A landscape of the new dark silicon design regime. *IEEE Micro, 33*(5), 8–19.