Worst-Case Guarantees on a Processor with Temperature-Based Feedback Control of Speed

PRATYUSH KUMAR and LOTHAR THIELE, ETH Zurich

On-chip temperatures continue to rise, in spite of design efforts towards more efficient cooling and novel low-power technologies. Run-time thermal management techniques, such as speed scaling and system throttling, constitute a standard component in today's processors. One such technique is the feedback control of the processing speed based on the on-chip temperature. If suitably designed, such a controller can ensure that the temperature of the processor does not exceed a given bound, independent of the application. Such isolation of needs is encouraging. However, from the application's stand-point, such a processor must provide performance guarantees; in particular, the guarantee that real-time jobs do not have worst-case delays larger than their relative deadlines. For applications which exhibit variability, such as bursty arrival patterns, computing such guarantees is not apparent. As key enablers in such a computation, for the specific setting of First-Come-First-Serve (FCFS) scheduling, we (a) define and prove a monotonicity principle satisfied by the processor with the said controller, and (b) propose a thermally clipped processor model. We identify the worst-case trace simulating which on a suitably chosen thermally clipped processor provides the tight upper-bound on the worst-case delay. These results hold for general models of (a) the power consumption of the processor, (b) its thermal model, (c) the speed scaling law, and (d) the task model. For this modelling scope, we show that the same worst-case trace also leads to the worst-case temperature of the processor. This is useful to characterise tasks which do not load the processor sufficiently to hit the given peak temperature bound. We demonstrate the utility of this calculation by designing a shaper to delay the arrival times of jobs and thereby restrict the observed worst-case temperature while still meeting the task's deadlines.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: *Real-time and embedded systems*

General Terms: Design, Performance, Reliability

Additional Key Words and Phrases: Speed scaling, dynamic thermal management, real-time systems, peak temperature analysis, traffic shaping

ACM Reference Format:
Pratyush Kumar and Lothar Thiele. 2014. Worst-case guarantees on a processor with temperature-based feedback control of speed. ACM Trans. Embedd. Comput. Syst. 13, 4s, Article 122 (March 2014), 26 pages.
DOI: http://dx.doi.org/10.1145/2584611

1. INTRODUCTION

Increasing functional requirements and rising leakage currents lead to ever-larger power densities in today's computing systems. This directly translates to escalating on-chip temperatures. Such high temperatures potentially degrade the performance and shorten the lifespan of the computing system [Skadron et al. 2004]. Experts have forewarned that a large proportion of the cost of future chips would have to be devoted to face up to this "temperature wall" [Borkar 1999; Tiwari et al. 1998].

This work is supported by the EU FP7 projects EURETILE and PRO3D, under grant numbers 247846 and 249776, respectively.
Authors' address: Computer Engineering and Networks Laboratory, ETH Zurich, CH-8092 Zurich, Switzerland; email: {pratyush.kumar, lothar.thiele}@tik.ee.ethz.ch.

DOI: http://dx.doi.org/10.1145/2584611

Research into new materials and technology innovations continue to reduce the power consumption. Simultaneously, chip packagers are exploring more effective cooling solutions such as liquid cooling [Brunschwiler et al. 2010]. In spite of these efforts, it is felt that run-time strategies be employed to reduce on-chip temperatures. For instance, in Dynamic Frequency Scaling (DFS) [Brooks and Martonosi 2001] the frequency or the processing speed of a processor is reduced when it gets too hot. Other examples of such Dynamic Thermal Management (DTM) techniques include system throttling [Skadron et al. 2004], load balancing [Sharma et al. 2005] and admission control [Kim et al. 2007]. Indeed, modern processors have such features built-in. For instance, the AMD Opteron 2218 series processor has five P-states with performance varying up to 2.6× [Dell 2007]. The P-states with higher frequency can execute tasks faster, but consume more power. Similarly, Intel Sandy Bridge processors have the Turbo Boost feature, which can run the processor at up to 1.2-1.3× the Thermal Design Power (TDP) [Naveh et al. 2011].

A concern with the use of such DTM techniques is the performance guaranteed by the system to a software task executing on it. In particular, a hard real-time application executing on a DTM-enabled processor must be guaranteed to meet its deadlines. The underlying question thus is: "How do we design and analyze DTM techniques with both temperature *and* performance constraints?"

Several research works have addressed this problem for the case of speed scaling. More specifically, researchers have proposed *co-design* where the speed scaling policy is designed to explicitly meet *both* temperature and performance constraints for given models of the architecture and application. A predictive technique to improve performance of multimedia applications was proposed in Srinivasan and Adve [2003]. The problem of maximising the resource utilisation in the presence of thermal constraints was studied in Rao et al. [2006] and Chantem et al. [2009]. The first work in considering thermal constraints in the presence of real-time requirements was initiated in Bansal et al. [2004]. In this work, speed scaling is refined online in response to arriving jobs. Chen et al. [2009], propose a proactive speed scaling technique that meets the tasks deadlines under thermal constraints. Zhang and Chatha [2007], provide competitive approximation algorithms for the computationally hard problem of thermal-aware scheduling. Speed scaling with precedence constraints has been studied in Pruhs et al. [2008]. Reducing of peak temperature while meeting end-to-end deadlines of task graphs is studied in Kumar and Thiele [2011b].

The fundamental limitation with this approach and similar approaches is that the DTM policy is verified for a *precise* model of the system. Given the nature of the problem, this model spans widely; it comprises the physical properties of heat diffusion, architectural properties of power generation and the software properties of jobs' arrival and execution times. It is not uncommon to expect these properties of a system to be uncertain at design-time; VLSI engineers have often highlighted process variations, whereas real-time systems routinely exhibit variable execution times and arrival patterns.

Such variability can prove to be detrimental; if, due to a software bug, the execution time of a task is larger than expected, the temperature constraint can be violated. This can lead to a physical failure. This motivates a design approach where we guarantee bounds on the different properties of the processor in *isolation*. For instance, it is desirable that the guaranteed peak temperature is independent of the application the processor is running. An example of such isolation, from a different area, is the use of reservation schemes to temporally isolate tasks sharing resources [Buttazzo 1997]. Indeed, such isolation is supported in some processors. For instance, the Intel Sandy Bridge processors allow the Turbo Boost only for managed short intervals of time up to

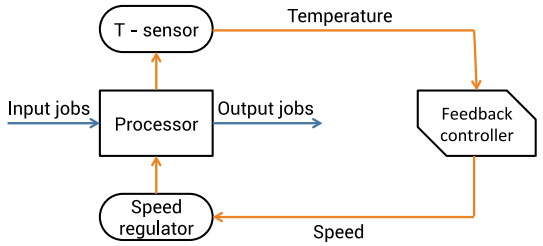


Fig. 1. Block diagram of a processor with temperature-based feedback controlled speed scaling.

30–60s [Naveh et al. 2011]. After this, the temperatures are too high and Turbo Boost is disabled independently of the executing application.

The reactive speed scaling approach proposed in Wang and Bettati [2008] provides an elegant method to isolate the peak temperature guarantee from the application properties. The authors proposed a two-speed scaling law that runs the processor in a slower speed when the temperature reaches a threshold. We extend this model significantly. We consider a speed scaling law with arbitrary number of speeds and a general feedback controller. We visualise such a processor in Figure 1. The processor serves a stream of jobs, and is equipped with a temperature sensor and a speed regulator which are interfaced with a feedback controller. In our proposed approach, the speed of the processor is varied as a feedback control law governed by the temperature of the processor: lower speeds at higher temperatures. If the feedback control law is suitably designed (cf. Section 2.2), an upper-bound on the temperature of the processor can be derived independent of the executing application.

An apparent disadvantage of such a processor is the difficulty in analysing the performance guaranteed by it; the speed at which the processor runs depends on its temperature, which in turn is a function of earlier jobs. For a given deterministic trace of jobs (arrival times and execution demands), simulation can evaluate the performance. The problem is more involved in the realistic setting of variable job arrival times and execution demands, where we would like to compute the worst-case delay. Consider, for instance, a periodic task that can infrequently exhibit jitter or has varying execution demands of jobs. For such streams, checking if deadlines are met is not straightforward. Wang et al. [2010], consider the problem of analysing the worst-case delay for the reactive speed scaling model they proposed in Wang and Bettati [2008]. Apart from the limitations in the modelling scope, the authors proposed computation technique requires solving of several non-linear constraints in an optimisation problem of uncharacterised size. Furthermore, the authors do not distinguish between different starting temperatures, which we show is crucial in the guarantees computed. Finally, the authors do not guarantee a tight bound on the worst-case temperature.

In this work, we consider a stream of jobs with variable arrival patterns and execution demands, as characterised by an arrival curve [Boudec and Thiran 2001]. This representation has been shown to model different kinds of variability expected in real-time systems [Thiele et al. 2000]. As the main contribution of this work, we present an analysis technique to compute the tight upper-bounds on both the delay and temperature when a stream of jobs satisfying the arrival curve characterisation is executed according to First-Come-First-Serve (FCFS) policy. As key enablers for this analysis, we (a) prove useful monotonicity principles satisfied on the considered processor, and (b) introduce the hypothetical model of a thermally clipped processor. Importantly, our results hold under relaxed assumptions on the power consumption pattern, the thermal model, the feedback control law, and the task model. We apply the proposed techniques to analyse guarantees for a stream of jobs that are preprocessed by a shaper.

The rest of the article is organised as follows. In Section 2, we formally define the system models and illustrate the working of the processor on an example set-up in Section 3. We consider timing analysis for different initial temperatures of the processor. For the case of maximum initial temperature we present a direct result in Section 4. We prove useful monotonicity principles in Section 5 and apply it to the case of minimum initial temperature. We then define a thermally clipped processor in Section 6 and use it for timing analysis for any initial temperature. In Section 7, we compute the tight bound on the worst-case temperature for a given task model. We analyse the preprocessing of jobs by a shaper in Section 8. We demonstrate the results in experimental simulations in Section 9, and conclude in Section 10.

2. SYSTEM MODEL

In this section, we will establish the models for the different components in the system. During modelling, we aim to retain as much generality as possible to enable an axiomatic presentation of the analysis.

2.1. Processor Model

We denote the processor with feedback controlled speed scaling as $\mathbf{P}$. Let $s(t)$, $P(t)$ and $T(t)$ denote the speed, power consumption and temperature of $\mathbf{P}$ at time t, respectively.

2.1.1. Available Speeds. The speed assigned to $\mathbf{P}$ must belong to a given set of allowed speeds denoted as $\mathbf{S}$. We consider both discrete and continuous speed assignment. For instance, we may allow all speeds in the interval [100MHz, 200MHz] or specific speeds in {100MHz, 150MHz, 200MHz}.

2.1.2. Power as a Function of Speed. The power consumption is a function of the speed of the processor and is given by the relation $P(t) = \phi(s(t))$

Assumption 1. The function ϕ is positive, convex and increasing.

Within this assumption, the function ϕ can model static power (usually constant), dynamic power (usually s^γ with $\gamma \in [2, 3]$) and leakage power that is constant or linearly dependent on the temperature of $\mathbf{P}$ (cf. A.2. of Appendix A).

2.1.3. Thermal Model. The thermal model of $\mathbf{P}$ models how the power consumption translates into heat and how the generated heat propagates within the processor and on to the ambient.

Assumption 2. The thermal model is linear and time-invariant (LTI).

This assumption holds for commonly adopted passive cooling solutions such as resistive heat sinks (cf. A.1 of Appendix A). Newer methods such as liquid cooling with phase transitions or propagation techniques such as radiation cannot be modelled.

For an LTI thermal model, the temperature is obtained as a convolution of the power trace with the impulse response. Let $h(t)$ denote the impulse response. Then we have

$$T(t) = T(0) + P(t) * h(t), \tag{1}$$

where $*$ is the standard convolution operator defined as

$$(f * g)(t) = \int_0^t f(u)g(t-u)du. \tag{2}$$

Assumption 3. The impulse response $h(t)$ is monotonically non-increasing.

The physical interpretation of this assumption is that the effect of power consumption on the temperature of the processor decreases over time, as the heat dissipates to the cooler surroundings. While this seems intuitively true, we formally show in Section A.2 of Appendix A, that it holds for several systems of interest. For instance, for the commonly used lumped compact thermal model, the impulse response exponentially decays.

Note that we do not characterise the thermal model in terms of specific thermal parameters such as heat conductances and capacitances. Instead, we use the impulse response formulation under the mild assumption of monotonicity. This illustrates the generic set of assumptions under which the subsequent results hold.

2.2. Speed Scaling Law

The speed scaling law is the relation used by the feedback controller to set the speed of $\mathbf{P}$ as a function of the current temperature.

Assumption 4. The delay between sensing of the temperature and actuation of the speed is negligible in comparison to the thermal rate constant.

Typical thermal rate constants are in the order of hundreds of milliseconds. We assume that the feedback control loop has a delay much smaller than this. Under this assumption, the speed scaling law is given by some function f such that

$$s(t) = f(T(t)). \tag{3}$$

The co-domain of f should be a subset of the set of allowed speeds $\mathbf{S}$. We allow both continuous and piecewise constant functions for f.

Assumption 5. The speed scaling law f is monotonically nonincreasing.

This assumption considers a class of intuitive laws, wherein, when the processor is getting hotter, the controller is only allowed to reduce the speed of the processor.

As we motivated, we would like to guarantee an upper-bound on the temperature, independent of the application. Let this be denoted as $T^{\max}$. Let the slowest speed within the co-domain of f be denoted as $s^{\min}$ and let $T^{\infty}(s^{\min})$ denote the steady-state temperature when running at $s^{\min}$.

Assumption 6. The speed scaling law f is such that $T^{\infty}(s^{\min}) = T^{\max}$.

Clearly, we cannot allow $T^{\infty}(s^{\min}) > T^{\max}$ as this violates the upper bound on the temperature. On the other hand, if we had $T^{\infty}(s^{\min}) < T^{\max}$, then the temperature will never reach $T^{\max}$. In this case, we are not sufficiently utilising the processing power while keeping the temperature below $T^{\max}$. Thus, we assume that the smallest speed in the co-domain of f is related as previously mentioned.

Wang and Bettati [2008], proposed reactive speed scaling under all the Assumptions 4, 5, and 6. In addition to these assumptions, they only considered two speeds and discrete speed scaling; assumptions which we do not make.

When there are no pending jobs to execute, the speed of $\mathbf{P}$ is considered to be zero. Power at zero speed $\phi(0)$ can be nonzero, thereby modelling idle power consumption.

2.3. Task Model

The task model characterises the stream of jobs that will execute on $\mathbf{P}$. In this work, we assume that jobs are executed in First-Come-First-Serve (FCFS) manner, that is, they are executed in the order in which they enter the pending queue. The task is characterized by the execution demand of the jobs: the number of processor cycles required to complete the job on $\mathbf{P}$. This execution demand would depend on the speed of the processor. In the presentation of results, we assume the simplified case where the execution demand scales linearly with speed: half the execution demand at twice

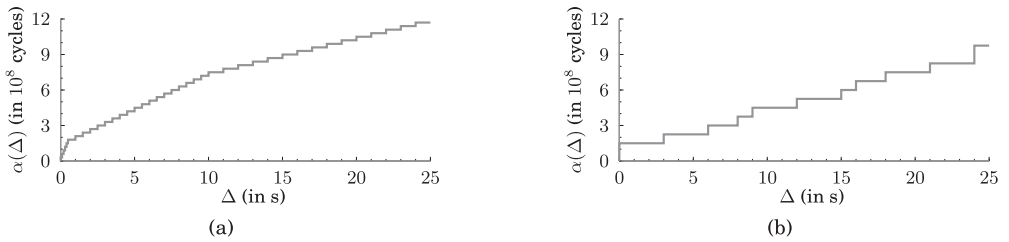


Fig. 2. Task models of two example tasks. The arrival curves are in terms of processor cycles.

the speed. However, as we increase the speed, we may not get a linear reduction in the execution demand (due to bandwidth or memory bottlenecks). We show in Section A.3. in Appendix A, that such a case can be considered within our framework with identical results.

As discussed earlier, variability in the arrival and execution times of jobs are expected and need to be considered. For instance, a periodic job may exhibit the jitter at unknown times. To formally capture such variability, the notion of arrival curve has been used in Network Calculus [Boudec and Thiran 2001], and subsequently Real-Time Calculus [Thiele et al. 2000]. We introduce this modelling framework.

For one trace of job arrivals, let $R(t)$ denote the cumulative execution demand of the jobs arriving in the interval $[0, t)$, that is, all jobs arriving until time t require a total of $R(t)$ processor cycles on $\mathbf{P}$. Let $\mathbf{R}$ denote the set of possible traces $R(t)$ that can arrive. Then, this set of traces $\mathbf{R}$ is characterised by the arrival curve α given by

$$\alpha(\Delta) = \sup_{t \geq 0}(R(t + \Delta) - R(t)), \quad \forall\, R \in \mathbf{R}. \tag{4}$$

In other words, $\alpha(\Delta)$ upper-bounds the cumulative execution demand of all jobs arriving in any time interval of length Δ, for any of the possible traces.

An arrival curve can model different job arrival patterns such as periodic, periodic with jitter, bursty, or sporadic patterns. As examples, we consider the two arrival curves that we shall consider later in the article. First, consider a bursty task that is characterised by three possible rates of job arrivals. Such tasks are common in the networking domain [Tanenbaum 2003] and can be represented using the leaky-bucket model with a buffer term for each rate. In the considered example, we have three rates of 1/s, 2/s, and 10/s, with buffer capacities of 15, 5, and 1, respectively, in units of 10^8 processor cycles. The worst-case execution time of each job is 0.3×10^8 processor cycles. The arrival curve that models the execution demand of such a stream of jobs is given in Figure 2(a). Second, consider a stream of jobs obtained by merging two periodic streams with periods 3s and 8s and execution demand of 0.75×10^8 processor cycles for each job. The arrival curve for this trace of jobs is shown in Figure 2(b).

Note that the modelling of arrival curves suffice to characterise the task in terms of computing the worst-case delay and temperature. In this sense, the modelling burden remains the same as in the analysis of real-time behavior of embedded systems.

2.4. Ambient and Initial Temperatures

The temperature and hence the speed of $\mathbf{P}$ depend on the ambient temperature, denoted as $T_{\mathtt{amb}}$. In existing work, a constant ambient temperature is assumed. Following a similar approach, we fix the ambient temperature to its largest possible value.

The delay of jobs and temperature of $\mathbf{P}$ also depend on the initial temperature of $\mathbf{P}$, denoted as $T(0)$. Jobs starting to execute on a hotter processor can potentially suffer

$$\mathbf{S} = [100, 200]\text{MHz}$$
$$\phi(s) = 2 + 12.5 \times \left(\frac{s}{100\text{MHz}}\right)^{2.3} \text{W}$$
$$h(t) = \exp(-0.25 \times t)\text{W/Ks}$$
$$T_{\text{amb}} = 292\text{K}$$
$$T^{\min} = 300\text{K}$$

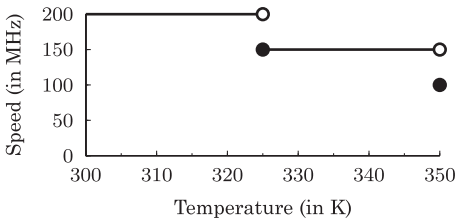


Fig. 3. Example model of a processor and the speed scaling law f.

longer delays. We thus consider the initial temperature as part of the input description. In the results, we explicitly consider different cases for different initial temperatures.

2.5. Observation Horizon

The problem specification also provides a time horizon of interest, denoted as τ. This horizon denotes the length of time, starting from the initial time, until which jobs of the task can arrive. The worst-case bounds on delay and temperature are to be identified for traces of job arrivals of length up to τ. Thus, if the initial time is taken as 0, the time horizon of job arrivals is $[0, \tau]$.

2.6. Problem Statement

Given are (a) $\mathbf{P}$ characterised by the set of allowed speeds $\mathbf{S}$, the relation between speed and power consumption ϕ, and the thermal impulse response h, (b) the speed scaling law f, (c) the arrival curve of the task model α, (d) the upper-bound on the ambient temperature T_{amb}, (e) the initial temperature $T(0)$, and (f) an observation horizon τ. We are to consider the execution of all streams of jobs that confirm to α with jobs arriving in the interval $[0, \tau]$. For all such streams, we are to compute (a) the worst-case delay suffered by the jobs, and (b) the worst-case temperature of $\mathbf{P}$.

3. ILLUSTRATING EXAMPLE

In Figure 3, we describe an example model with power consumption as a function of speed, thermal impulse response, ambient temperature and the speed scaling law. The assumptions hold: ϕ is convex and increasing, h is monotonically decreasing, and f is monotonically nonincreasing. The processor model is such that the minimum temperature of the processor in the idle state is $T^{\min} = 300K$. This includes the effect of the ambient temperature and the idle power consumption at the steady state. The speed scaling law has the slowest speed $s^{\min} = 100\text{MHz}$. The steady-state temperature at this speed is 350K, which indeed equals $f(s^{\min})$ for the considered speed scaling law. Thus, the considered model represents a valid model with temperature of the processor varying in the range $[300, 350]$K.

We now consider the execution of a specific stream of jobs on this processor starting from an initial temperature $T(0) = 310\text{K}$. Let two jobs J_1 and J_2 with execution demands of 3×10^8 cycles and 1×10^8 cycles, respectively, arrive at times 0 and 6s, respectively. For this specific trace of job arrival patterns, we plot in Figure 4(a) the evolution of the system in terms of the processor temperature and speed. J_1 starts executing at 0 and completes execution at 2.1s, while J_2 starts at 6s and finishes at 6.6s. Notice the effect of the speed scaling law. At low temperature, for instance in the interval $[0, 0.2]$s, $\mathbf{P}$ runs at the fast speed of 200MHz. After reaching the threshold of 325K, the speed is regulated down to 150MHz. Similarly, on reaching the threshold of 350K, the speed is regulated down to 100MHz. When running at this speed ($= s^{\min}$), the speed of the processor stays at 350K ($= T^{\max}$). In the absence of pending jobs, the temperature decreases due to the much lesser idle power. As a future reference

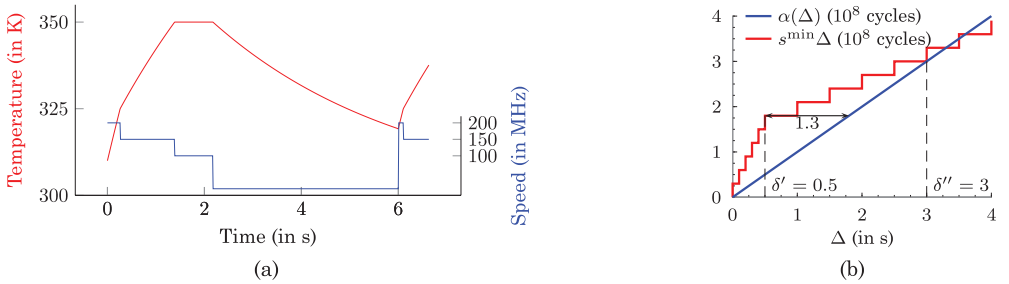


Fig. 4. (a) Illustration of the working of a processor with feedback-controlled speed scaling. (b) Computation of worst-case delay starting with a maximum initial temperature.

(cf. Section 5), the finish time of job J_2 is 6.6s and the temperature at that time is 338K, respectively.

We discuss the practical challenges in implementing the described temperature management policy, primarily the need to modify the speed based on the current temperature. Processors are equipped to completely power-down if the temperature exceeds a certain specified bound. To implement this scheme, this must be extended to have multiple such *temperature interrupts* for the different temperatures in the speed scaling law f. For instance, the `lmsensors` library, available on Unix operating systems, can be programmed to generate interrupts when the temperature exceeds certain bounds. Clearly, having more points in f can lead to more frequent speed changes which may have an overhead. Trading off the smoothness of the temperature control with the overhead is a key design choice.

4. DELAY ANALYSIS FOR MAXIMUM INITIAL TEMPERATURE

We first consider the worst-case delay computation when $\mathbf{P}$ starts with the maximum initial temperature, that is, $T(0) = T^{\max}$. Recall that $T^{\max}$ is an upper-bound on the temperature of the processor for any application. Starting from this high temperature, the jobs are likely to suffer longer delays. We would like to tightly bound such delays.

First, we define `Del` as used in Network Calculus Boudec and Thiran [2001]

$$\texttt{Del}(f, g, \tau) = \sup_{t \in [0,\tau]} \{\inf\{u \in [t, \tau] \ : \ f(t) \leq g(t+u)\}\}. \tag{5}$$

In words, $\texttt{Del}(f, g, \tau)$ defines the maximum horizontal distance between functions f and g, in the domain $[0, \tau]$. In the following result, we show that by a simple application of the `Del` function, we can compute the desired tight worst-case delay for the specific case of maximum initial temperature.

THEOREM 4.1. *The tight worst-case delay when a stream of jobs with input arrival rate α is executed on $\mathbf{P}$ with $T(0) = T^{\max}$ is given by*

$$d = \texttt{Del}(\alpha(\Delta),\ s^{\min}\Delta,\ \tau). \tag{6}$$

PROOF. We will first show that d as defined here is an upper-bound on the worst-case delay under the given conditions. We will then show the tightness by producing a valid trace of jobs, $R^*_{\max}$, such that at least one job suffers a delay exactly equal to d.

Recall that $s^{\min}$ is the lowest speed of $\mathbf{P}$ when there are pending jobs to execute, and $f(T^{\max}) = s^{\min}$. Then, it follows that in any busy interval of length Δ, the minimum service provided by $\mathbf{P}$ is $(s^{min} \cdot \Delta)$. From results in Network Calculus [Boudec and Thiran 2001], we know that the service curve of $\mathbf{P}$ is lower-bounded by $\beta(\Delta) = s^{min} \cdot \Delta$. From the definition of the service curve, the delay suffered by any job of the stream is indeed upper-bounded by d defined as in (6).

We now prove the tightness. Let δ' and δ'' be the smallest numbers satisfying

$$\alpha(\delta') = s^{min}(\delta' + d), \tag{7}$$

$$\alpha(\delta'') = s^{min}\delta''. \tag{8}$$

Then, it follows from the subadditive property of α [Boudec and Thiran 2001] that $\delta' \leq \delta''$. Consider the trace of jobs $R^*_{\max}$ defined as

$$R^*_{\max}(t) = \alpha(t), \quad t \in [0, \min(\tau, \delta'')] \tag{9}$$

For this trace, the definitions of δ' and δ'' imply that if the jobs of $R^*_{\max}$ are executed on a constant-speed processor executing at speed $s^{\min}$, we have a busy interval from $[0, \delta'']$ and the job arriving at δ' (which is before the end of the first busy interval) suffers the largest delay which equals d.

Now consider the execution of $R^*_{\max}$ on $\mathbf{P}$. Since, the initial temperature is $T^{\max}$, $\mathbf{P}$ will behave exactly like a constant-speed processor running at $s^{\min}$ until the end of the first busy interval. From the result that $\delta' \leq \delta''$, we know that all jobs of $R^*_{\max}$ belong to the first busy interval, and thus $\mathbf{P}$ behaves exactly like a constant-speed processor running at s^{min}. Thus, some job of $R^*_{\max}$ suffers the delay d, and the bound is tight. □

We illustrate this analysis with an example. Consider the processor model and speed scaling law of Figure 3 and the task model of Figure 2(a). To compute the worst-case delay for the maximum initial temperature, we have to compute the `Del` function between the arrival curve and the straight line through origin with the slope of the slowest speed. This is shown in Figure 4(b). The worst-case delay is obtained to be 1.3s.

We discussed earlier the possible variability in the job arrival pattern and the apparent difficulty in computing the worst-case delay under feedback-controlled speed scaling. This result, however, provides a direct technique to tightly compute the worst-case delay across all such variabilities. Indeed, the worst-case trace of execution demand is one that is equal to the arrival curve α. Note that when starting from the highest temperature possible, the worst-case delay depends only on the slowest speed. It does not depend on the other speeds, the power consumption, the thermal model or the speed scaling law. As we will see later in Section 9, the worst-case delay is larger for larger initial temperatures. Thus, across all initial temperatures, the worst-case delay suffered by a stream of jobs, is given by the simple relation of (6).

5. MONOTONICITY PRINCIPLES AND APPLICATION TO DELAY ANALYSIS WITH MINIMUM STARTING TEMPERATURE

In this section, we will establish some useful monotonicity principles exhibited by the processor. Subsequently, we will use these principles to compute the worst-case delay for the minimum starting temperature.

5.1. Monotonicity Principle

First, we define a *unit cycle*, denoted as u, as the smallest measure of the execution time of jobs. Conveniently, u can be defined to be one cycle long. We assume that the speed of $\mathbf{P}$ is constant during the execution of a unit cycle.

We now present some results on the execution of a single unit cycle on $\mathbf{P}$. These results will be used to analyse execution of any given pattern of job arrivals.

LEMMA 5.1. *Consider the execution of a unit cycle on* $\mathbf{P}$. *Increasing the starting temperature of* $\mathbf{P}$ *cannot lead to an earlier finish time of the unit cycle.*

PROOF. The speed-scaling rule f is monotonically nonincreasing with the temperature of the processor. Thus, increasing the temperature cannot increase the speed of execution of the unit cycle and thus cannot decrease the finish time. □

LEMMA 5.2. *Consider the execution of a unit cycle on* $\mathbf{P}$. *Delaying the execution of this unit cycle cannot lead to an earlier finish time.*

PROOF. We prove this by contradiction. Consider two schedules S_1 and S_2. Both schedules start at time 0 with the same arbitrary initial temperature of $\mathbf{P}$. S_1 executes the unit cycle from time 0, whereas S_2 is idle until time $p > 0$ and then executes the unit cycle. Let T_1 and T_2 denote the temperature trace of $\mathbf{P}$ for the schedules S_1 and S_2, respectively. Let P_1 and P_2 denote the power traces of $\mathbf{P}$ for schedules S_1 and S_2, respectively. Let the finish time of the unit cycle be earlier in S_2 than in S_1. Then, there exists an earliest time $q > p$ such that both S_1 and S_2 have completed exactly the same amount of work until q. The two schedules perform the same amount of work in $[0, q]$ but schedule S_1 runs at a lower speed. From the convexity of function ϕ, we know that

$$\int_0^q P_1(t)dt \leq \int_0^q P_2(t)dt. \tag{10}$$

Furthermore, P_1 is constant, whereas P_2 is nondecreasing. Combining this with the monotonically decreasing impulse response h, we have

$$(P_1 * h)(q) \leq (P_2 * h)(q). \tag{11}$$

Since the initial temperature of the $\mathbf{P}$ for the two schedules is the same, we have $T_1(q) \leq T_2(q)$. However, at time q, the speed of $\mathbf{P}$ in S_2 is greater than in schedule S_1. Given the monotonicity of f, we have $T_2(q) < T_1(q)$. This is a contradiction and hence S_2 does not finish executing the unit task before S_1 does. □

LEMMA 5.3. *Consider the execution of a single unit cycle on* $\mathbf{P}$. *Delaying the execution of this unit cycle cannot lead to a lower temperature of* $\mathbf{P}$ *measured at the finish time of the unit cycle in the delayed execution.*

PROOF. Let S_1 and S_2 be two schedules defined in the proof of Lemma 5.2. Let d_1, d_2 be the finish times of the unit task in schedules S_1 and S_2, respectively. Then, we need to show that $T_1(d_2) \leq T_2(d_2)$. In S_1, $\mathbf{P}$ runs at a constant speed from time 0 to d_1 and then is idle from d_1 to d_2. In S_2, $\mathbf{P}$ is idle from 0 to q and then runs at a higher constant speed from time q to d_2. The two schedules complete the same amount of work in the interval $[0, d_2]$ and schedule S_1 runs at a lower speed. Then, from the convexity of function ϕ, we know that

$$\int_0^{d_2} P_1(t)dt \leq \int_0^{d_2} P_2(t)dt. \tag{12}$$

Further, P_1 is monotonically nonincreasing, whereas P_2 is monotonically nondecreasing. As before, using the monotonic natures of the impulse response h and the speed-scaling rule f, we have $T_1(d_2) \leq T_2(d_2)$. □

Note that the convexity of ϕ, the monotonicity of functions f and h, are the only properties that suffice to prove these results. In particular, one does not need the exact solution of the thermal differential equation. Given this set of properties on the execution of the unit cycle, we are now ready to define and derive the monotonicity principle that holds for the execution of any job arrival pattern.

THEOREM 5.4 (MONOTONICITY PRINCIPLE). *Consider a* given *stream of jobs served by* $\mathbf{P}$. *Delaying the arrival time of any job of the stream cannot lead to an earlier finish time of any* subsequent *job of the stream.*

PROOF. We prove this by induction on the total number of unit cycles of all jobs of the considered stream.

Induction Hypothesis. Delaying the execution of any job or increasing the starting temperature of **P** will not lead to an earlier finish time of any job.

Basis. For one unit cycle, Lemma 5.1 and 5.2 prove the induction hypothesis.

Inductive Step. If the hypothesis holds for all streams with execution demand c unit cycles, then it holds for all streams with execution demand $(c + 1)$ unit cycles.

Let R be a stream of jobs with a total execution demand of $(c + 1)$ unit cycles. Let S_1 and S_2 be two schedules of executing R on **P** such that in S_1 all jobs are executed without any delay, whereas in S_2 some jobs are executed with delay. Consider the first unit cycle of R. If the execution of this unit cycle is not delayed in S_2, then both schedules S_1 and S_2 behave exactly similarly until the execution of the first unit cycle and thus the problem is reduced to a stream of tasks of c unit cycles which, according to the inductive step satisfies the hypothesis. Let the execution of the first unit cycle be delayed in S_2. Let d_1 and d_2 denote the finish time of this unit cycle in the two schedules S_1 and S_2. From Lemma 3, we know that $d_2 \geq d_1$. Two cases arise, depending on the arrival time of the second unit cycle of R, denoted as η.

Case (a). $\eta \geq d_2$. Applying Lemma 4 to S_1 and S_2 for the fixed-time of observation of η, we have, $T_1(\eta) \leq T_2(\eta)$. Consider the trace R starting from η onwards. The execution demand of this trace is c cycles. Then, according to the induction hypothesis increasing the starting temperature cannot lead to the earlier finish time of any job. As $T_1(\eta) \leq T_2(\eta)$, no job of S_2 can finish before the same job of S_1.

Case (b). $\eta < d_2$. Modify S_1 to form a new schedule S_1' such that the second unit cycle of R is delayed to start executing from d_2. From the induction hypothesis, from the second unit cycle onwards, no job of schedule S_1 will finish any later than the same job in S_1'. From the induction hypothesis, starting from time d_2, no job of S_2 can finish before the same job from S_1'. Hence, no job of S_2 finishes before the same job of S_1. □

To put this result in perspective, consider the case of a constant-speed processor serving a stream of jobs in first-come-first-serve (FCFS) manner. It can be seen that delaying the execution of any job will not lead to a earlier finish time of any subsequent job. This can be interpreted as: Delaying a job does not decrease the interference that job has on subsequent jobs. This derived results says that, even for processors with temperature-controlled speed scaling, under the considered model assumptions, this desirable monotonic relationship between arrival time and interference holds. The satisfaction of this rule indicates that in spite of the apparent complexity of the processor with feedback-controlled speed scaling, intuitive monotonicity principles still hold. This motivates the existence of simple analysis techniques.

We illustrate the monotonicity principle using the example chosen in Figure 4(a). Recall that in the specific trace considered there were two jobs J_1 and J_2 arriving at times 0 and 6s, respectively. Both jobs were executed without any delay upon their arrival. Let us now delay the execution of J_1 to be start from 3s. The evolution of the processor's temperature and speed for this execution is shown in Figure 5. According to the result, the later execution of J_1 will lead to larger finish time of job J_2 and increase the temperature in at that time. This is indeed shown to be true by the plot. Specifically, finish time of job J_2 of 6.8s and finish temperature of 350K, are both larger than the corresponding values in Figure 4(a).

5.2. Worst-Case Delay for Minimum Initial Temperature

The lowest temperature of the processor is obtained when there is no prior workload. In this case, the temperature is given by $T^{\min}$ which is the steady-state temperature when the processor is idle. We now study computing the worst-case delay on the processor starting with this minimum initial temperature, that is, $T(0) = T^{\min}$. In the

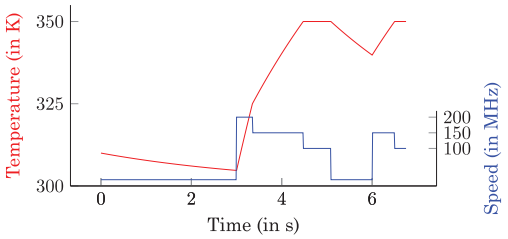


Fig. 5. Illustration of the monotonicity principle.

subsequent result, using the monotonicity principle, we show that independent of the thermal, power and speed-scaling parameters of $\mathbf{P}$, there exists a uniquely characterized worst-case trace and a uniquely characterized job of that trace which experiences the worst-case delay. Recall that input traces are defined in the domain $[0, \tau]$, for a given observation horizon τ.

THEOREM 5.5. *The tight worst-case delay when a stream of jobs with arrival curve α is executed on $\mathbf{P}$ with $T(0) = 0$, is suffered by the last job of the trace R^* defined as*

$$R^*(t) = \alpha(\tau) - \alpha(\tau - t), \quad t \in [0, \tau]. \tag{13}$$

PROOF. Let the last job of the trace R^* be denoted J^*. We prove the result by contradiction. Let there exist some other trace of arrival of jobs, R, such that the job arriving at time $t'(\leq \tau)$, denoted as J', has a response time larger than J^*. Let t'' be defined as

$$R(t') = \alpha(t' - t''). \tag{14}$$

Since R conforms to the input rate α, we have $t'' \in [0, t']$.

Applying the monotonicity principle on R, we can delay the arrival of jobs arriving in $[0, t']$, without decreasing the response time of J'. This can be done until

$$\begin{aligned} R(t) &= \alpha(t' - t'') - \alpha(t' - t), \quad t \geq t'', \\ &= 0, \qquad\qquad\qquad\quad t < t''. \end{aligned} \tag{15}$$

It can be seen that delaying any job any further results in a trace that does not conform to the input rate α. The modified trace R defined as described here in the interval $[t'', t']$ is a suffix of the trace R^* defined in (13). More formally,

$$R(t) = R^*(t + (\tau - t')), \quad t'' \leq t \leq t'. \tag{16}$$

Let us compare the common parts of the traces, that is, $R(t'' : t')$ and $R^*((\tau - (t' - t'')) : \tau)$. While serving trace R, the buffer of pending jobs is empty just before t'', whereas while serving R^* there could be pending jobs at time $\tau - (t' - t'')$. Then, from Lemma 5.2, for J' to suffer a larger delay than J^*, the temperature of $\mathbf{P}$ while executing R at time t'' should be strictly higher than the temperature of $\mathbf{P}$ while executing R^* at time $\tau - (t' - t'')$. However, this is a contradiction, as the temperature of $\mathbf{P}$ while executing R remains the minimum possible value of 0 in $[0, t'']$.

R^* conforms to the input arrival rate α and thus the result is tight. □

Note that under the model assumptions, the worst-case trace R^* does not depend on the values of parameters of the speed-scaling rule, the power consumption or the thermal impulse response. To compute the value of the worst-case delay, we simply simulate the execution of R^* on $\mathbf{P}$ and observe the delay of the last job.

We illustrate this computation on an example problem. Consider the processor model and speed scaling law of Figure 3, and the task model of Figure 2(a). The worst-case trace of jobs for this system, $R^*(t)$ for an observation horizon of 25s, is shown

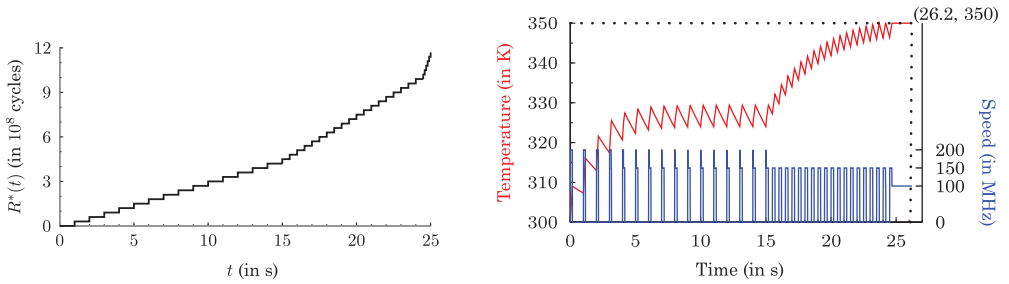


Fig. 6. Computation of the worst-case delay for minimum initial temperature. (a) Worst-case trace of job arrivals, and (b) evolution of processor temperature and speed for this trace.

in Figure 6(a). Notice that the bursts of jobs arrive at the end of the observation horizon. The execution of this worst-case trace on the considered processor is shown in Figure 6(b). The worst-case delay is indeed suffered by the last job and equals 1.2s.

Note that in R^* the burst of jobs, if any, appears at the end of the time horizon. The worst-case trace is such that it attempts to maximise the interference that the other jobs induce on the last job of the trace, by saving the burst of jobs till the end of the observation horizon.

6. A THERMALLY CLIPPED PROCESSOR AND APPLICATION TO DELAY ANALYSIS FOR ANY INITIAL TEMPERATURE

We presented the results of delay analysis for the case of maximum and minimum initial temperatures. In both cases, we applied different principles that enabled identification of the worst-case trace. So the natural question is: what happens when the initial temperature is between the two extremes. We answer this question and develop a unifying analysis technique in this section.

Towards, this end, we first define a model of a hypothetical processor called a thermally clipped processor, which we denote as **P'**. We will show, in the subsequent result, that analysis of worst-case traces on **P'** helps provide a unifying computation technique for delay analysis on the original processor **P**.

A thermally clipped processor **P'** has a speed scaling rule, power model, and thermal model exactly like that of **P**. But **P'** is differentiated from **P** by an additional rule which states that the temperature of **P'** is never allowed to fall below a given temperature threshold. Note that we do not change the thermal model: the temperature of the system is computed according to the standard thermal and power models. However, only if the temperature is below the threshold, it is enforced to be equal to the threshold. In other words, the temperature of **P'** is *clipped* from below to the threshold.

Let us consider some intuitive examples of **P'**. A thermally clipped processor **P'** with threshold temperature equal to the minimum temperature of **P**, is identical to the original processor **P**. A thermally clipped processor **P'** with temperature threshold equal to the maximum temperature of **P** is equivalent to a constant-speed processor running at the slowest speed $s^{\min}$.

In terms of the thermally clipped processor, we now present an analysis technique to compute the tight worst-case delay for an intermediate starting temperature.

THEOREM 6.1. *The tight worst-case delay when a stream of jobs with arrival curve α is executed on* **P** *with $T(0) = T^{\text{inter}}$, for some given $T^{\text{inter}} \in (0, T^{\max})$, is equal to the delay suffered by the last job of the trace R^* defined in (13) when served by* **P'** *with temperature threshold equal to T^{inter}.*

PROOF. Let the last job of R^* be denoted as J^*.

We first prove that the upper-bound holds by contradiction. Let there be some other stream R executing on $\mathbf{P}$ such that the job J' arriving at t', suffers a delay larger than that suffered by J^* when served by $\mathbf{P'}$. As before, we can use the monotonicity principle to delay the arrival of jobs without decreasing the delay suffered by J'. With such delays, we can modify R until

$$\begin{aligned} R(t) &= R(t') - \alpha(t' - t), \quad t \geq t'', \\ &= 0, \qquad\qquad\qquad\quad t < t'', \end{aligned} \tag{17}$$

where t'' is as defined in (14).

Now we can compare the two equal parts of the traces R and R^*, namely $R(t'' : t')$ and $R^*(\tau - (t' - t'') : \tau)$. There are no pending jobs at time t'', when R is served by $\mathbf{P}$. Whereas there may be pending jobs when R^* is served by $\mathbf{P'}$ at time $\tau - (t' - t'')$. By repeated use of Theorem 5.2 and the definition of $\mathbf{P'}$, for the hypothesis to hold, the temperature of $\mathbf{P}$ at time t'' when serving R, must be strictly larger than that of $\mathbf{P'}$ at time $\tau - (t' - t'')$ where serving R^*. But $\mathbf{P}$ is idle from in $[0, t'']$ and thus its temperature cannot be larger than T^{inter} which is the lower-bound on the temperature of $\mathbf{P'}$. Thus, we have a contradiction.

Since the delay is computed on an hypothetical processor $\mathbf{P'}$, we still have to show that the bound is tight when jobs execute on the original processor $\mathbf{P}$. Let us consider the trace R^* being executed by $\mathbf{P'}$. Let ρ be the latest time when the temperature of $\mathbf{P'}$ is *forced* to be equal to T^{inter}. If the temperature is never forced to be equal to T^{inter}, then $\mathbf{P'}$ behaves equivalent to $\mathbf{P}$, and thus the bound is tight, as R^* conforms to α. In this case, we set $\rho = 0$. Otherwise, $\rho > 0$. Since, the temperature of $\mathbf{P'}$ at ρ^- cannot be lower than T^{inter}, it implies that there are no pending jobs to execute at time ρ. Thus, starting from an empty buffer at ρ, $\mathbf{P'}$ executes all jobs that arrive subsequently, without forcing the temperature to T^{inter}. This implies that if we have a trace R^*_{inter} defined as follows then the behavior of $\mathbf{P}$ and $\mathbf{P'}$ when executing R^*_{inter} is identical.

$$R^*_{\text{inter}}(t) = R^*(t + \rho) - R^*(\rho), \qquad t \in [0, \tau - \rho]. \tag{18}$$

$$= \alpha(\tau - \rho) - \alpha(\tau - \rho - t), \qquad t \in [0, \tau - \rho]. \tag{19}$$

Thus, the delay suffered by the last job of R^*_{inter} on $\mathbf{P}$ is the same as the delay suffered by the last job of R^* on $\mathbf{P'}$. This proves the tightness. □

This result provides an elegant approach to compute worst-case delay for any intermediate temperature. We simply simulate the trace R^*, obtained by flipping the arrival curve in the observation horizon, on the hypothetical processor $\mathbf{P'}$ with the threshold temperature equal to the initial temperature. The delay suffered by the last job in this simulation is the tight worst-case delay.

We illustrate this for an example problem. Consider the processor model and speed-scaling law of Figure 3, and the task model of Figure 2(b). Let us compute the worst-case delay starting from the initial temperature of 330K. The execution of the worst-case trace on the hypothetical processor $\mathbf{P'}$ with temperature threshold 330K is shown in Figure 7. Notice the several points where the temperature of the processor is clipped to the threshold of 330K. The worst-case delay is indeed suffered by the last job and equals 1.4s. The last point where the temperature is clipped to 330K is 50s. Thus, the worst-case trace for the original processor $\mathbf{P}$ starting from the initial temperature of 330K is the arrival of the two jobs at time 0. The higher the initial temperature; the shorter will be the worst-case trace on the original processor.

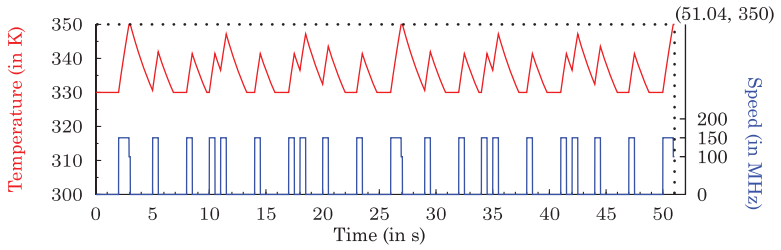


Fig. 7. Computation of worst-case delay on a thermally clipped processor with temperature threshold 330K.

6.1. Unifying Analysis for Any Initial Temperature

Using the hypothetical processor **P'**, we analysed the worst-case delay for the case of an intermediate initial temperature. Can we apply the same methodology to the two cases of maximum and minimum temperature?

Extending the analysis to consider the case of minimum initial temperature is straightforward. In this case, **P'** has a threshold temperature equal to the minimum temperature of **P** and is indistinguishable from **P**. Thus, the latest time when the temperature is clipped, denoted as ρ, would be 0. Then, from (18), we know that R^*_{inter} and R^* are equal. Effectively, we would simulate the worst-case trace R^* on **P**. According to Theorem 5.5, the last job of this trace indeed gives the worst-case delay.

Can a similar statement be made for the case maximum initial temperature? For this case, **P'** has a threshold temperature of $T^{\max}$, which implies that **P'** always executes jobs at the constant speed of $s^{\min}$. Then, it can be shown that the value ρ would be exactly equal to $\max(0, \tau - \Delta'')$, where Δ'' is as defined in (8). We then obtain R^*_{inter} by substituting this ρ in (19). It can be shown that the delay suffered by the last job of such an R^*_{inter} when served at a constant speed $s^{\min}$ is equal to d as defined in (6). Thus, the analysis with thermally clipped processors can be extended to consider both minimum and maximum initial temperatures. This provides a unifying analysis technique across all considered model variations.

As has been illustrated, for different initial temperatures, the worst-case traces could be different, depending on the value of ρ. However, by defining the hypothetical model **P'**, we need to only consider a single worst-case trace, namely R^*. The delay of the last job of this trace when served by **P'** is a tight upper-bound for the delay of any job when executing on **P**, independent of the initial temperature. We have thus effectively reached our goal of handling variations in the software model by only analysing one trace to compute the worst-case delay. Importantly, this trace is independent of the properties of the processor.

Finally, we would like to highlight the design principle that we can derive from this analysis. As a direct consequence of the monotonicity principle, for all initial temperatures, the worst-case delay of a stream of jobs is monotonic with respect to the input arrival rate. For instance, if there are two software designs where the input arrival rates are α_1 and α_2, respectively, with $\alpha_1 > \alpha_2$, then independent of the parameters of **P** or the initial temperature, the worst-case delay of α_1 cannot be smaller than that of α_2. This clear separation of principles satisfyingly reinforces the common idea that bursty output must be avoided, if possible.

7. ANALYSIS OF WORST-CASE TEMPERATURE

In the earlier sections, we analysed the worst-case delay suffered by a stream of jobs when executing on a processor with temperature-controlled speed scaling. Such a processor, by definition, provides an upper-bound on the worst-case temperature, namely $T^{\max}$, which is the steady-state temperature of the processor when running at the

slowest set speed of $s^{\min}$. However, certain applications may not stress the processor sufficiently to reach this upper-bound. For such applications, it is desirable to compute a tighter bound on worst-case temperature than $T^{\max}$.

Like in the case of delay analysis, we will use two key enablers in the analysis of the worst-case temperature: (a) a monotonicity principle and (b) the use of simulation on a thermally clipped processor. We will show using these results that for the same worst-case trace that leads to the worst-case delay of the jobs also leads to the worst-case temperature. Moreover, this worst-case temperature is observed at the end of the execution of the job that suffers the worst-case delay.

THEOREM 7.1. *Consider a stream of jobs being served by* **P**. *Delaying the execution of any job cannot lead to a lower temperature at any subsequent time.*

PROOF. Let the last job of the trace R^* be denoted J^*.

We prove the result by contradiction. The temperature of **P** rises during the execution of a job. Thus, the worst-case temperature must be at the end of execution of some job. We call the temperature at the end of execution of a job, its finish temperature. Let there exist some other trace of arrival of jobs, R, such that the finish temperature of the job arriving at time $t'(\leq \tau)$, denoted as J', be larger than the finish temperature of J in R^*. Let t'' be defined as follows

$$R(t') = \alpha(t' - t''). \tag{20}$$

Since R conforms to the input rate α, we have $t'' \in [0, t']$.

Applying the monotonicity principle of Lemma 5.3 on R, we can delay the arrival of jobs arriving in $[0, t']$, without decreasing the finish temperature of job J'. This can be done until

$$\begin{aligned} R(t) &= \alpha(t' - t'') - \alpha(t' - t), \quad t \geq t'', \\ &= 0, \qquad\qquad\qquad\quad t < t''. \end{aligned} \tag{21}$$

It can be seen that delaying any job any further results in a trace that does not conform to the input rate α. The modified trace R as defined here in the interval $[t'', t']$ is a suffix of the trace R^* defined in (13). More formally,

$$R(t) = R^*(t + (\tau - t')), \quad t'' \leq t \leq t'. \tag{22}$$

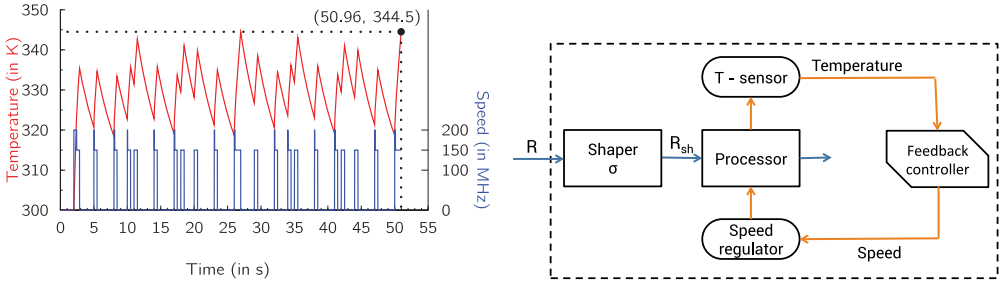
Let us compare the common parts of the traces, that is, $R(t'' : t')$ and $R^*((\tau - (t' - t'')) : \tau)$. While serving trace R, the buffer of pending jobs is empty just before t'', whereas while serving R^*, there could be pending jobs at time $\tau - (t' - t'')$. Then, from Lemma 5.2, for J' to have a higher finish temperature than J^*, the temperature of **P** while executing R at time t'' should be strictly higher than the temperature of **P** while executing R^* at time $\tau - (t' - t'')$. However, this is a contradiction, as the temperature of **P** while executing R remains the minimum possible value of 0 in $[0, t'']$.

R^* conforms to the input arrival rate α and thus the result is tight. □

This result mirrors the similar result of Theorem 5.4 for the computation of worst-case delay. This result can then be used to compute the worst-case trace for analysing the worst-case temperature, using simulation on the thermally clipped processor. We present this in the subsequent result.

THEOREM 7.2. *The tight worst-case temperature when a stream of jobs with arrival curve α is executed on* **P** *with initial temperature $T(0) \in [0, T^{\max}]$, is equal to the temperature at the end of the execution of the last job of the trace R^* defined in (13) when served by* **P'** *with temperature threshold equal to $T(0)$.*

PROOF. Let the last job of R^* be denoted as J^*.



(a) (b)

Fig. 8. (a) Worst-case temperature computation. (b) Preprocessing stream of jobs with a shaper.

We first prove that the upper-bound holds by contradiction. Let there be some other stream R executing on $\mathbf{P}$ such that the job J' arriving at t', has a finish temperature larger than the proposed worst case. As before, we can use the monotonicity principle to delay the arrival of jobs in the trace R, without decreasing the finish temperature of J'. With such delays we can modify R until

$$\begin{aligned} R(t) &= R(t') - \alpha(t' - t), \quad t \geq t'', \\ &= 0, \quad t < t'', \end{aligned} \tag{23}$$

where t'' is as defined in (14).

Now we can compare the two equal parts of the traces R and R^*, namely $R(t'' : t')$ and $R^*(\tau - (t' - t'') : \tau)$. There are no pending jobs at time t'', when R is served by $\mathbf{P}$. Whereas there may be pending jobs when R^* is served by $\mathbf{P}$' at time $\tau - (t' - t'')$. By repeated use of Theorem 5.2 and the definition of $\mathbf{P}$', for the hypothesis to hold, the temperature of $\mathbf{P}$ at time t'' when serving R, must be strictly larger than that of $\mathbf{P}$' at time $\tau - (t' - t'')$ where serving R^*. But $\mathbf{P}$ is idle from in $[0, t'']$ and thus its temperature cannot be larger than $T(0)$ which is the lower-bound on the temperature of $\mathbf{P}$'. Thus, we have a contradiction.

Like in the proof of Theorem 6.1, we can identify the worst-case trace that leads to this peak temperature on $\mathbf{P}$, in terms of ρ the latest time when the temperature of the processor is clipped in the simulation of R^*. This proves the tightness. □

From this result, we know that the same worst-case trace when simulated on a thermally-clipped processor with threshold temperature equal to the initial temperature, leads to the worst-case delay and worst-case temperature. Furthermore, these worst-case properties are exhibited at the same time, that is, at the finish of the last job of the trace. In addition, this shows the design principle of making the arrival curve less bursty, simultaneously decreases the worst-case delay and the worst-case temperature.

We illustrate the above computation with an example. For the processor model and speed scaling law of Figure 3 and the task model of Figure 2(b), we computed the worst-case temperature starting from $T^{\min}$ as the initial temperature. By simulating the trace R^*, we indeed obtain the worst-case temperature of 344.5 K ($<T^{\max}$) at the end of the last job as shown in Figure 8(a).

8. COOL SHAPERS WITH FEEDBACK CONTROLLED SPEED SCALING

We have seen that the bursty nature of a variable arrival pattern plays a role in the worst-case delay and temperature. A burstier arrival pattern can lead to both larger delay and temperature. In yet another context, such arrival patterns are detrimental: in computer networks, bursty arrival patterns usually lead to large delays and

requirement of large buffers. Hence, *shapers* are used to limit the rate of incoming packets to given patterns [Boudec and Thiran 2001]. More specifically, delays are inserted between packets to ensure that at the output of the shaper the rate of arriving packets conforms to a given arrival curve.

For the specific case of reducing peak temperature within timing constraints, we proposed design of optimal shapers in Kumar and Thiele [2011a]. The technique was applied to constant speed processors, where by delaying the jobs we could effectively throttle the system. With a good choice of the shaper, we can reduce the peak temperature while meeting all deadlines of tasks.

Can a similar technique be used for processors with feedback-controlled speed scaling? Consider the block diagram of Figure 8(b). A stream of jobs is first processed by a shaper and then the jobs are executed on **P**. We are interested in analysing the worst-case total delay of the jobs (across both the shaper and **P**) and the worst-case temperature of **P**.

We first begin with a description of shapers and some known results. A shaper is characterised by a shaping curve, denoted as σ, where $\sigma(\Delta)$ upper-bounds the execution demand of the stream of jobs arriving in any interval of length Δ, at the output of the shaper. This execution demand, like in the case of arrival curve, can be specified in terms of processor cycles. From known results in Network Calculus [Boudec and Thiran 2001], for a given input stream of jobs with arrival curve $R(t)$, the shaped output stream of jobs has execution demand $R_{\text{sh}}(t)$ given by

$$R_{\text{sh}} = R \otimes \sigma, \tag{24}$$

where

$$(f \otimes g)(\Delta) = \min_{0 \le \lambda \le \Delta} (f(\lambda) + g(\Delta - \lambda)). \tag{25}$$

Similarly, the arrival curve at the output of the shaper, α_{sh}, is related to the arrival curve of the input, α as

$$\alpha_{\text{sh}} = \alpha \otimes \sigma. \tag{26}$$

When a stream of jobs with arrival curve α is shaped by a shaping curve σ, the worst-case delay suffered by any job, denoted d_{sh}, is given by

$$d_{\text{sh}} = \texttt{Del}(\alpha, \sigma, \infty), \tag{27}$$

where `Del` is as defined in (5).

What is it that a shaper can potentially do? Can it minimise both the worst-case temperature and delay? We answer this in the subsequent results.

THEOREM 8.1. *By introducing a shaper, for a given input task, the worst-case temperature of* **P** *cannot increase.*

PROOF. By introducing a shaper, the effective arrival curve of the stream of jobs at **P** is given by α_{sh} as defined in (26). From the definition of the $\otimes$ operator, we have

$$\alpha_{\text{sh}} \not\succ \alpha. \tag{28}$$

From the derived monotonicity principle, we know that a smaller arrival curve cannot lead to a larger worst-case temperature. □

THEOREM 8.2. *By introducing a shaper, the worst-case delay (across both the shaper and* **P***) of jobs cannot decrease.*

PROOF. From the monotonicity principle of Theorem 5.4, we know that delaying the execution of a job cannot lead to an earlier finish for any subsequent job. The shaper

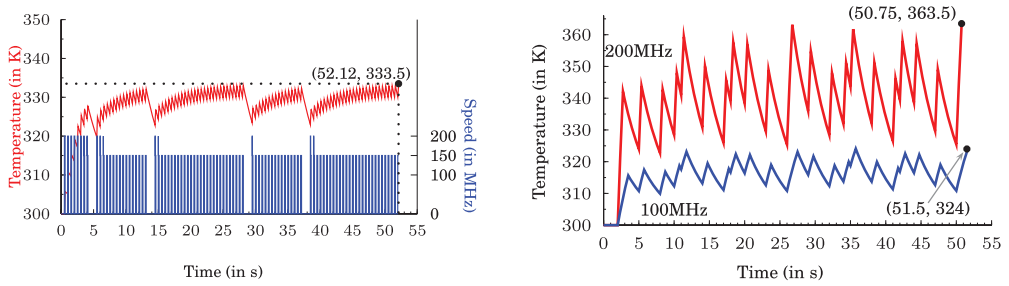


Fig. 9. (a) Computation of worst-case parameters in the presence of a shaper. (b) Computation of worst-case parameters for two constant speed processors.

delays the execution of different jobs on $\mathbf{P}$. For any trace of job arrivals, such shaping cannot lead to a smaller delay than the execution without shaping. □

From these results, we know that the shaper can be used to potentially decrease the worst-case temperature. But the worst-case delay cannot be decreased by introducing the shaper. In other words, the minimum worst-case delay is experienced when we greedily execute jobs without introducing any delay.

We illustrate the usage of a shaper for an example. Consider the processor model and speed scaling of Figure 3 and the task model of Figure 2(b). Starting with the initial minimum temperature, the worst-case delay and temperature are obtained to be 0.96s and 344K, respectively (cf. Figure 8(a)). We now introduce a shaper with a periodic shaping curve of period 0.5s and allowed execution demand in one period of 0.19×10^8cycles. When the input stream of tasks is shaped by this shaper, the worst-case delay and temperature, as obtained from Figure 9(a), are 2.12s and 333.5K. Indeed, the worst-case temperature is reduced at the cost of an increased worst-case delay. Notice how the introduction of the shaper "slices" the execution of the jobs into small pieces (in this case of 0.19×10^8 cycles).

9. EXPERIMENTAL RESULTS

In this section, we will present experimental results that evaluate different aspects of a processor with feedback controlled speed scaling. We will illustrate the dependence of the worst-case delay and temperature on different parameters such as the observation horizon, the initial temperature, the speed scaling law, the choice of shapers and errors in the sensor.

9.1. Advantage of Feedback Controlled Speed Scaling

We first illustrate the advantage of feedback controlled speed scaling. Consider the processor model of Figure 3 and the task model of Figure 2(b). We choose a observation horizon of 50s. Let the initial temperature of the processor equal $T^{\min}$. We are required to meet performance and temperature constraints. Let the relative deadline of the jobs be 1s and the maximum allowed temperature be 350K.

As a first option, consider executing the jobs at the constant highest possible speed, namely $s = 200$MHz. Since a constant speed processor is a special case of feedback controlled speed scaling, we can apply the results developed here to this setting. Thus, the worst-case delay and temperature are obtained by simulating the trace R^* as shown in Figure 9(b). The worst-case delay is 0.75s and the worst-case temperature is 363.5K. The jobs meet their deadline but the peak temperature exceeds the threshold of 350K.

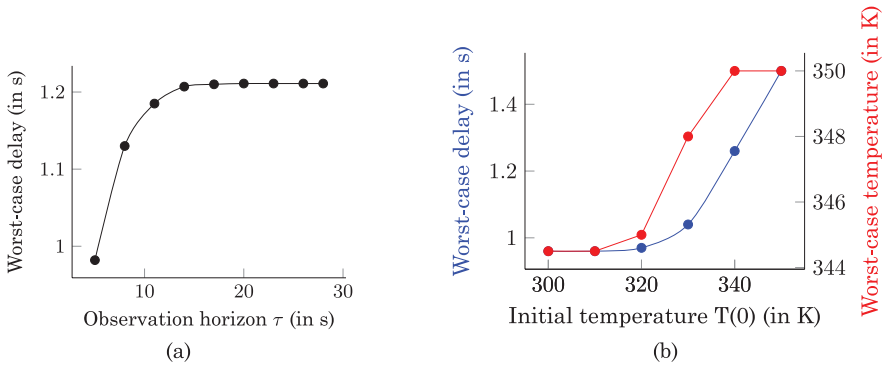


Fig. 10. Variation of worst-case parameters in terms of (a) observation horizon, and (b) initial temperature.

We try to rectify breaching of the temperature constraint. The steady-state temperature of the processor when executing at speed 100MHz is 350K. Thus, executing at this speed will meet the temperature constraint. As the second option, consider executing the jobs at the constant lowest speed of $s = 100$MHz. The worst-case delay and temperature are obtained by simulating the worst-case trace R^* as shown in Figure 9(b). The worst-case delay is 1.5s and the worst-case temperature is 324K. While the temperature constraint is met, we are unable to meet the timing constraint.

Thus, with the two considered constant speed schedules, we are unable to meet both the constraints simultaneously. Now, consider the use of the speed scaling law of Figure 3. We illustrated the computation of the worst-case delay and temperature in Figure 8(a). The worst-case delay is 0.96s and the worst-case temperature is 344.5K, thereby meeting both the constraints. In addition, the temperature constraint is met independent of the timing properties.

This example highlights the advantage of the feedback control. On the one hand, it slows the processor as it gets hot, thereby meeting temperature constraints. On the other hand, it speeds up the processor to reduce the finish time of jobs, thereby meeting deadlines.

9.2. The Choice of the Observation Horizon

The observation horizon denoted as τ is the length of the interval of time over which we observe the arrival of jobs. In our work, we have considered it as a given parameter. We now explore how such a horizon could be chosen if it is not known.

The general solution technique we have proposed simulates the behavior of the processor for jobs arriving within the horizon. Thus, larger the value of τ, longer is the computation cost of the analysis. However, with too small a value of τ, we may compute erroneous bounds on the worst-case delay and/or temperature. We do not expect to identify an analytical technique to compute such a threshold, as our computation is based on simulating a worst-case trace. Instead, we propose analysing the system for different values of τ.

We perform such an analysis for the processor model and speed scaling model of Figure 3 and the task model of Figure 2(a). We plot the variation of the computed worst-case delay with τ in Figure 10(a). From the plot, the value of the worst-case delay reaches a peak around 20s. Simulations on shorter observation horizons lead to erroneous bounds, while the higher cost of larger observation horizons does not increase accuracy. Thus, in our experiments, we have chosen a value of 25s.

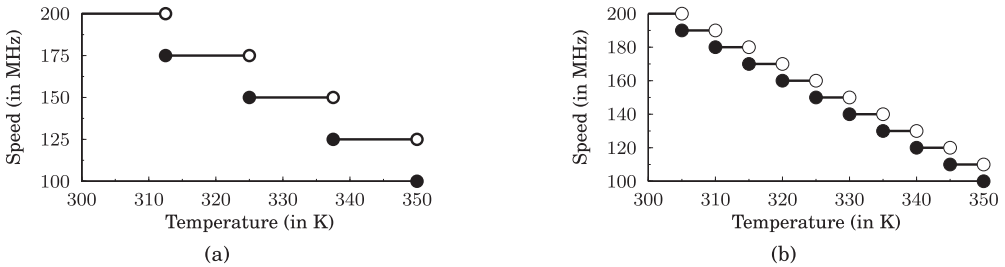


Fig. 11. Finer speed scaling laws.

9.3. Effect of Initial Temperature

By defining the thermally clipped processor, we identified a unified analysis step to compute worst-case delay and temperature, irrespective of the initial temperature. However, the values of the worst-case parameters are dependent on the initial temperature. By applying the monotonicity principles, we can show the following result.

LEMMA 9.1. *The worst-case delay and temperature when a stream of jobs executes on* **P** *are monotonically nondecreasing with the initial temperature* $T(0)$.

We illustrate this monotonic dependence for the processor and speed scaling models of Figure 3 and the task model of Figure 2(b) in Figure 10(b).

9.4. Finer Speed Scaling Laws

The speed scaling law can be any monotonically nonincreasing function We have considered the speed scaling law of Figure 3 with three discrete speeds uniformly distributed over the temperature range of 300 to 350 K. For the task model of Figure 3(a), with this speed scaling law, we obtained worst-case delay and temperature of 1.2s and 350K, respectively (cf. Figure 5). The speed scaling law can be made finer with additional speeds. Does such an intuitive refinement reduce the worst-case delay and temperature? We explore this for the task model of Figure 2(a).

Consider the speed scaling law of Figure 11(a). It has five discrete speeds uniformly distributed over the temperature range. The worst-case delay and temperature are obtained to be 1s and 350K, respectively. The worst-case delay is lower than that with the previous speed scaling law, without increasing worst-case temperature. Thus, it seems that refinement in this case improved the worst-case parameters. We consider a further refinement; the speed scaling law with 11 speeds in Figure 11(b). The worst-case delay and temperature are obtained to be 1.06s and 348.3K, respectively. Though the peak temperature is lesser, this additional step of refinement has increased the worst-case delay.

In conclusion, having a larger number of speed levels uniformly distributed over the temperature range, does not necessarily lead to lower worst-case metrics. The right spacing amongst these speeds is crucial.

9.5. Application-Dependence of Optimal Speed Scaling Law

We continue our experimentation on the effect of the speed scaling law on the worst-case delays and temperature. In this experiment, we will attempt to have a non-uniform speed scaling law. In particular, we will consider a speed scaling laws with three discrete speeds. Two of these are fixed, namely 100MHz and 200MHz. The intermediate speed

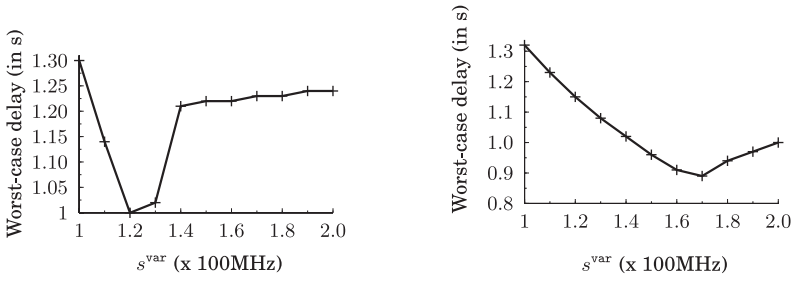


Fig. 12. Dependence of the worst-case delay on the value of a speed (s^{var}) in the speed scaling law for the task model of (a) Figure 2(a) and (b) Figure 2(b).

is a variable, which we would like to choose. We denote this speed as s^{var}. Formally, the speed scaling law is given as

$$\begin{aligned} f(T) &= 200 \text{ MHz}, && T < 325\text{K} \\ &= s^{\text{var}} \text{ MHz}, && 325\text{K} \leq T < 350\text{K} \\ &= 100 \text{ MHz}, && T = 350\text{K}. \end{aligned} \tag{29}$$

We compute the worst-case delay for different values of s^{var} for the task model of Figure 2(a). This variation is plotted in Figure 12(a). Note that there is a large variation, with a pronounced minimum, in the worst-case delay in relation to the value of s^{var}. Particularly, the relationship is neither monotonic nor convex.

The natural question now is whether a similar dependence exists for other task models. We repeat the experiment for the task model of Figure 2(b). The corresponding dependence of the worst-case delay on the value of s^{var} is shown in Figure 12(b). Again, there can be a substantial improvement in the worst-case delay with a judicious choice of s^{var}. However, the near-optimal setting (120 MHz) for the previous task model, performs relatively poorly for this task model. Thus, the right value of s^{var}, and by extension the speed scaling law, is dependent on the task model considered.

9.6. Choice of Shapers

We saw that shapers can be used to reduce the worst-case temperature at the potential cost of a larger worst-case execution time. However, there exists a design flexibility in choosing the shaping curve. In Figure 9(a), we analysed the task model of Figure 2(b) for a specific shaper. We repeat the analysis for different shaping curves. In each case we choose a periodic shaping curve of a given period, such that the average rate of execution demand at the output is no more than 0.375×10^8 cycles in 1s. The obtained variation is shown in Figure 13(a). As we reduce the period of the shaping curve both the worst-case parameters fall. The worst-case delay trends towards the limit of 2s, which is the worst-case delay inserted by the shaper. However, the period of the shaping curve cannot be infinitesimally reduced as switching on and off the processor can have a time or energy overhead which we do not model in this work.

9.7. Inaccurate Sensors

Often fabricating accurate temperature sensors and placing them at the desired part of the chip is error-prone. The common sensor errors are (a) offset: the sensed temperature is off by a some fixed value, (b) saturation: the sensed temperature does not increase beyond a threshold. Is the flipped trace still the worst-case trace in the presence of such errors?

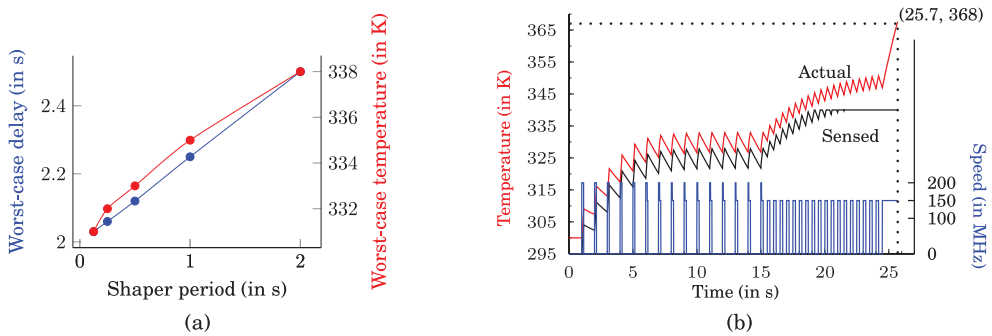


Fig. 13. (a) Variation of worst-case parameters for different shapers. (b) Worst-case flipped trace with an erroneous sensor.

We consider a faulty sensor with an offset of −5 K and a threshold of 340 K, the task model of Figure 2(b) and the processor model of Figure 3. We simulate the flipped trace and the resultant behaviour is shown in Figure 13(b). As the sensor under-estimates the temperature, we get a smaller worst-case delay (0.7 s vs 1.2 s in Figure 6(b)) and a larger worst-case (actual) temperature (368 K vs 350 K in Figure 6(b)). Indeed, as before, the worst-case delay is observed for the last job of the flipped trace.

We can generalise this observation to extend our results to a class of erroneous sensors as defined in the following lemma.

LEMMA 9.2. *Let $\epsilon(T)$ denote the sensed temperature when the actual temperature is T. If ϵ is a monotonically nondecreasing function, worst-case delay is given by the delay of the last job of the flipped trace.*

The intuition for the proof is to translate the error in the sensing into an modified speed scaling law. Since, the speed scaling law should satisfy Assumption 5, we have a requirement on the error model of the sensor. Given this result, even in the presence of sensor errors, we can execute the flipped trace on the system and identify the worst-case delay.

10. CONCLUSIONS AND FUTURE WORK

We presented the case for temperature-based feedback control of the speed of a processor to isolate the guarantee of meeting temperature constraints from the properties of the application. From the application's stand point, such a setup does not directly provide performance guarantees, in particular when the application can experience variability in execution demand. We considered this problem for a model that was kept generic by a minimal set of assumptions on the system parameters. For the FCFS scheduling, we showed that useful properties of monotonicity, expected in single-speed processors, are satisfied. We were then able to use these monotonicity principles to derive the worst-case pattern of job arrivals, which when simulated on a thermally-clipped processor leads to both the worst-case temperature and delay observed at the end of the execution of the last job. Importantly, the worst-case trace of jobs is independent of the processor parameters. We then studied the worst-case analysis for a stream of jobs which are preprocessed by a shaper. Such shapers can be used to decrease the peak temperature at the potential cost of increasing the worst-case delay. Finally, we studied the dependence of the worst-case parameters on several parameters. In particular, we showed how our results extend when sensors have errors.

Three directions of future work emerge. The first direction is aimed towards the design challenge of identifying the optimal speed scaling law for a given application. We

saw in the results that this the worst-case parameters are sensitive to the choice of the speed scaling law. The second direction is to extend these results to multicore systems where the power consumed on other cores also affect a given core, with nonmonotonic impulse responses. The final direction is to consider other scheduling policies amongst multiple streams of jobs.

APPENDIX A

A.1. Modelling Scope of Thermal Model

Recall that we consider a thermal model which is entirely described by the impulse response of the temperature for a unit power consumed, denoted as $h(t)$. We assume that this function is monotonically decreasing. We illustrate here how some commonly used thermal models can be formulated using the considered model.

A commonly used thermal model is the conductive heat-dissipation model, wherein the temperature of the processor evolves according to the following differential equation

$$GT(t) + C\frac{dT(t)}{dt} = P(t),$$

where G is the thermal conductance and C the thermal conductance. Then, it can be shown that this thermal behavior can be modelled by the monotonically decreasing impulse response $h(t) = \frac{1}{C}\exp(\frac{-t}{C/G})$.

Consider a more elaborate model of a cooling system as proposed in Huang et al. [2006]. Here the active heat-generating processor is interfaced with the ambient through several layers including substrates, pads, heat spreaders, and sinks. In such a system, the thermal model is a large RC network, where each node has a capacitance to the ground and nodes are interconnected with positive resistances. It can be shown that, for any such passive network of capacitances and conductances, the impulse response of the temperature of the heat-generating node is monotonic. Thus, as long as the temperature sensor is placed at the single heat-generating node, the model assumptions hold for nonlumped cooling systems.

A.2. Modelling Temperature-Dependent Leakage Power

In newer technologies, leakage power is a significant proportion of the total power consumption. Unfortunately, leakage power increases with temperature, sometimes leading to a run-off cycle. We now show that such a power consumption component can be modelled in our framework.

Let the total power consumption of the processor be given as $P(t) = \phi(s(t)) + \eta \cdot T(t) + \nu$, where ϕ is the convex function which denotes the dynamic power, while the leakage power is represented as a linear function of the temperature with the multiplier η. Then, the differential equation can be written as

$$(G - \eta)T(t) + C\frac{dT(t)}{dt} = \phi(s(t))\nu.$$

This is similar in form to the model of A.1. of the appendix. Thus, if the temperature-dependence of power consumption or thermal parameters can be approximated to be linear, the impulse response exists and is monotonically decreasing.

A.3. Nonideal Speed-Up of Execution Demand

The execution demand of a job may not scale linearly with the speed, due to bottlenecks from other microarchitectural units such as I/O, memory and cache. Let the execution demand of a job at any speed s be proportional to $c\chi(s)$ for some speed-up factor χ. Owing to the said bottlenecks, we expect $\chi(s)/s$ to be monotonically nonincreasing. Thus, at

higher speeds, we have a higher execution demand than the ideal assumption. This is exactly similar to the effect of the power consumption which is a convex increasing function. Thus, in Lemmas 5.1-5.3, a nonideal χ would further delay the finish time of a unity cycle if we delay its execution. We defer formal proofs due to lack of space.

REFERENCES

Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. 2004. Dynamic speed scaling to manage energy and temperature. In *Proceedings of FOCS*.

S. Borkar. 1999. Design challenges of technology scaling. *Micro. IEEE* 19, 4, 23–29.

Jean-Yves Le Boudec and Patrick Thiran. 2001. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Lecture Notes in Computer Science, Springer.

David Brooks and Margaret Martonosi. 2001. Dynamic thermal management for high-performance microprocessors. In *Proceedings of HPCA*. IEEE Computer Society, 171–182.

T. Brunschwiler, S. Paredes, U. Drechsler, B. Michel, W. Cesar, Y. Leblebici, B. Wunderle, and H. Reichl. 2010. Heat-removal performance scaling of interlayer cooled chip stacks. In *Proceedings of the 12th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm),* 1–12.

Giorgio C. Buttazzo. 1997. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Norwell, MA, USA.

Thidapat Chantem, Xiaobo Sharon Hu, and Robert P. Dick. 2009. Online work maximization under a peak temperature constraint. In *Proceedings of ISLPED*. Jörg Henkel, Ali Keshavarzi, Naehyuck Chang, and Tahir Ghani, Eds., ACM, 105–110.

Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. 2009. Proactive speed scheduling for real-time tasks under thermal constraints. In *Proceedings of RTAS*.

Dell. 2007. Managing Data Center Power and Cooling. http://www.dell.com/downloads/global/power/ps1q07-20070204-AMD.pdf.

Wei Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan. 2006. HotSpot: A compact thermal modeling methodology for early-stage VLSI design. *IEEE Trans. VLSI Syst.* 14, 5.

Kyong Hoon Kim, Rajkumar Buyya, and Jong Kim. 2007. Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters. In *Proceedings of CCGRID*. IEEE Computer Society, 541–548.

Pratyush Kumar and Lothar Thiele. 2011a. Cool shapers: shaping real-time tasks for improved thermal guarantees. In *Proceedings of DAC*. Leon Stok, Nikil D. Dutt, and Soha Hassoun, Eds., ACM, 468–473.

Pratyush Kumar and Lothar Thiele. 2011b. Thermally optimal stop-go scheduling of task graphs with real-time constraints. In *Proceedings of ASP-DAC*.

Alon Naveh, Doron Rajwan, Avinash Ananthakrishnan, and Eli Weissmann. 2011. Power management architecture of the 2nd generation Intel® Core microarchitecture, formerly codenamed Sandy Bridge. http://www.hotchips.org/wp-content/uploads/hc_archives/hc23/HC23.19.9-Desktop-CPUs/HC23.19.921.SandyBridge_power_10-Rotem-Intel.pdf.

Kirk Pruhs, Rob van Stee, and Patchrawat Uthaisombut. 2008. Speed scaling of tasks with precedence constraints. *Theory Comput. Syst.* 43, 1, 67–80.

Ravishankar Rao, Sarma B. K. Vrudhula, Chaitali Chakrabarti, and Naehyuck Chang. 2006. An optimal analytical solution for processor speed control with thermal constraints. In *Proceedings of ISLPED*. Wolfgang Nebel, Mircea R. Stan, Anand Raghunathan, Jörg Henkel, and Diana Marculescu, Eds., ACM, 292–297.

Ratnesh K. Sharma, Cullen Bash, Chandrakant D. Patel, Richard J. Friedrich, and Jeffrey S. Chase. 2005. Balance of power: Dynamic thermal management for internet data centers. *IEEE Internet Comput.* 9, 1, 42–49.

Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. 2004. Temperature-aware microarchitecture: Modeling and implementation. *Trans. Archit. Code Optium.* 1, 1, 94–125.

Jayanth Srinivasan and Sarita V. Adve. 2003. Predictive dynamic thermal management for multimedia applications. In *Proceedings of ICS*. Utpal Banerjee, Kyle Gallivan, and Antonio González, Eds., ACM, 109–120.

Andrew S. Tanenbaum. 2003. *Computer Networks.* 4th Ed. Prentice Hall, Page 400.

L. Thiele, S. Chakraborty, and M. Naedele. 2000. Real-time calculus for scheduling hard real-time systems. In *Proceedings of ISCAS*.

Vivek Tiwari, Deo Singh, Suresh Rajgopal, Gaurav Mehta, Rakesh Patel, and Franklin Baez. 1998. Reducing power in high-performance microprocessors. In *Proceedings of DAC*. 732–737.

Shengquan Wang, Youngwoo Ahn, and Riccardo Bettati. 2010. Schedulability analysis in hard real-time systems under thermal constraints. *Real-Time Syst.* 46, 2, 160–188.

Shengquan Wang and Riccardo Bettati. 2008. Reactive speed control in temperature-constrained real-time systems. *Real-Time Syst.* 39, 1–3.

Sushu Zhang and Karam S. Chatha. 2007. Approximation algorithm for the temperature-aware scheduling problem. In *Proceedings of ICCAD*.

Received June 2012; revised April 2013 and October 2013; accepted January 2014