Sustainable Computing: Informatics and Systems 1 (2011) 46–56

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Short survey

Temperature-aware computing

Israel Koren*, C.M. Krishna

Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, United States

ARTICLE INFO

Article history:
Received 12 October 2010
Received in revised form 26 October 2010
Accepted 27 October 2010

Keywords:
Temperature-aware
Power-aware
Hotspots
Thermal models
Floorplan
Voltage scaling
Run-time temperature management

ABSTRACT

Temperature-aware techniques have established themselves as crucial steps during the design and operation of new complex ICs (e.g. dual-core microprocessors) in order to protect the ICs against high temperatures that may drastically reduce their lifetime or even render them inoperable. These techniques have been developed after it became clear that power-aware techniques and low-power design are insufficient since they still allowed *hotspots* to develop in the chip with temperatures considerably higher than the average temperature.

The goal of this paper is to provide an overview of the state-of-the-art of temperature-aware computing. After a brief introduction, we present the current approaches to measuring the temperature of a circuit during its operation and to estimating, during the design phase, the maximum temperature that the circuit will experience. We then survey the known techniques for designing a chip with lower maximum temperature. This is followed by reviewing the currently employed run-time temperature management techniques.

This paper presents a thorough review of the research done in the past decade or so in the field of thermal-aware computing and lists most of the relevant journal and conference papers on this topic.

1. Introduction

Over the past two decades, as the performance of processors has steadily increased, their power consumption has increased considerably as well. The power density of a high-end microprocessor chip today (in terms of Watts dissipated per square area of surface) exceeds that of a hotplate. Power densities of up to 200 Watts/centimeter^2 are projected (see, for example [31]).

Two technical challenges arise due to this increase in power consumption. The first is the stress it places on batteries in portable devices such as laptops and phones. Here, the designer's aim is to control the overall power and energy consumption in order to lengthen battery life. Power- and energy-aware techniques such as voltage and frequency scaling have been developed to meet this challenge.

The second challenge is the thermal impact of power dissipation, and is the focus of this survey paper. High temperature greatly shortens the lifespan of a processor. It has been estimated that every 10 °C increase in temperature reduces component life by 50% [1]. Mechanisms such as electromigration and time-dependent dielectric breakdown increase exponentially with temperature [25]. Indeed, the time to failure has been shown to be a function of the form $\exp(-E_a/(kT))$, where E_a is the activation energy of the underlying failure mechanism, k is Boltzmann's constant, and T is the absolute temperature [1]. Even if an immediate failure does not result, detrimental permanent or intermittent changes can occur to the electrical characteristics of the device.

Heating due to power consumption also participates in an insidious positive feedback loop. Power consumption can be categorized into *dynamic* and *leakage* components. Dynamic power is consumed when devices switch from one logic level to another, and is related to the level of computational (switching) activity. Leakage power is power that flows from source to ground whenever a device is powered up, and does not contribute to the required operation of the device. In years past, the leakage component was but a small fraction of the overall power consumption. However, as feature sizes have dropped into the deep submicron range, this component has emerged to be on a par with the dynamic component. Leakage power grows exponentially with temperature. As a result, we have the following vicious cycle: the chip's temperature increases as it functions; this results in an increase in leakage power, which in its turn heats up the chip some more, and so on. Such an event, called "thermal runaway" can, if not stopped by some active power management technique, result in the failure of the chip.

One obvious approach to dealing with this problem is better cooling techniques. Heat sinks have become bigger over the years and air cooling is used (in desktop and laptop computers) to prevent them from over-heating, but there is a limit to air cooling's capability to dissipate heat. In general, cooling is expensive: designing

* Corresponding author. Tel.: +1 413 545 2643.
E-mail addresses: koren@ecs.umass.edu (I. Koren), krishna@ecs.umass.edu (C.M. Krishna).

2210-5379/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.suscom.2010.10.004

a system for the worst thermal case can add significantly to the cost of a computer system. It is estimated to cost about $3 per Watt to dissipate heat effectively [24]. For this reason, while heat sinks and fans are at the core of thermal management for high-powered systems, they must be complemented by a spectrum of additional techniques. It is these techniques that we survey in this paper.

It might be thought that power-aware techniques should be able to deal with the thermal impact of power dissipation as well. After all, it is due to power consumption that heating occurs. Unfortunately, power-aware techniques by themselves are not sufficient, since logic blocks within the chip have different power densities (due, for example, to very different levels of switching activity) and as a result, are not heated up uniformly. The thermal map of a chip often shows wide variations in temperature. For example, the cache may be cold while the register file is very hot. For this reason, the *average* temperature of a chip, which can be obtained by analyzing its overall power consumption, cannot by itself be a sufficient indicator of a thermal crisis. Furthermore, localized heating occurs much faster than chip-wide heating and therefore monitoring the rate at which the average temperature increases cannot provide timely indication of a developing hotspot. In addition, the specific blocks that develop hotspots may change from one application (executing on the processor) to another or even from one computation phase of an application to another. In general, power-aware techniques have only a limited effect on the maximum temperature of a logic block in a chip since they do not reduce the power density of the hottest blocks.

Thermal-aware techniques span a wide range. At the design stage of the chip, its floorplan can be set up in such a way as to minimize the number and extent of hotspots. For example, one might place subsystems which give off a lot of heat close to others that generally do not. This can be supplemented by effective approaches to managing overheating as and when it occurs. One can place several temperature sensors throughout the chip to monitor the temperature of its subsystems. If a subsystem becomes too hot, it can be throttled back to allow it to cool. A processor may reduce the level of aggressiveness in its pre-fetching as it approaches critical temperature levels. In multicore systems, tasks can be migrated from one core to another. Sophisticated techniques can be employed to estimate the thermal profiles of executing processes. Such profiles can be used by the operating system in its scheduling. For instance, one might give hotter processes smaller time-slices than cooler ones; alternatively (or additionally), one can ensure that hot processes are not scheduled one after the other but are separated by cooler ones to reduce the chances of overheating. Also, chip-wide voltage-scaling techniques (originally developed to render systems power-aware) can be used.

The remainder of this paper is organized as follows. In Section 2, we describe techniques for thermal measurement with on-chip sensors or performance counters as well as theoretical ways by which to model heat flow, a necessary component of schemes to project the temperature of individual blocks within a designed chip. We follow this up in Section 3 with a description of thermal-aware techniques in chip design. In Section 4, we outline runtime thermal-aware management techniques, for use when the chip is in operation. The paper concludes with a brief summary in Section 5.

2. Thermal measurement and modeling

To facilitate thermal management, we need techniques by which temperatures can be measured while the chip is in operation. We also require models of heat flow that can be used to efficiently predict, during the design phase, the expected temperature of the different chip blocks in order to modify the design if necessary. These temperature projection models can be calibrated using on-chip temperature sensors.

2.1. Temperature sensing

There are two ways in which to sense the temperature on a chip: directly by means of temperature sensors and indirectly through the use of performance counters. Clearly, a direct measurement by an accurate sensor can be expected to provide better information than an indirect estimate based on performance counters. However, the former also has a few limitations: it may be too expensive to populate the chip with a sufficiently high density of such sensors, the sensor's response (to changes in the chip temperature) time may be too large, its precision may be insufficient, and the sensor itself can contribute to further heating of the chip as a result of its own power dissipation [48]. Furthermore, temperature sensors can take milliseconds to access and obtain their readings in some cases [45].

Temperature sensors exploit the fact that a large number of electrical parameters of solid-state devices are temperature-dependent [57]. One way to estimate the temperature is by measuring the discharge time of a capacitor through leakage current. The subthreshold leakage current I_{leak} is approximately equal to:

$$I_{leak}(T) = K_a e^{-K_b/T}\,(1 - e^{-qV_{DS}/kT}) \tag{1}$$

where T is the temperature, K_a and K_b are constants, q is the charge of an electron, V_{DS} is the drain–source voltage, and the gate-source voltage is zero [40]. We can estimate this current by measuring the time it takes to discharge a known capacitance. This can be done by connecting the output of the capacitor to an inverter and charging the capacitor by means of a pulse. It is then discharged by the leakage current of a transistor. When the capacitor voltage falls below the switching threshold of the inverter, the inverter switches. The duration between the input pulse to the capacitor and the transition instant of the inverter is a measure of the discharge time, and thus, indirectly, of the temperature. The accuracy of this approach depends, to a large extent, on the accuracy with which the discharge time is measured.

Another approach exploits the fact that the difference in voltage across two diodes depends on their respective diode areas, their current ratios and the absolute temperature [75]:

$$V_{d1} - V_{d2} = \frac{KT}{q}\ln\left(\frac{I_1 A_2}{I_2 A_1}\right) \tag{2}$$

where V_{di} is the voltage across diode i $(i = 1, 2)$, A_i is the area of diode i, I_i the current flowing through it and K is a constant.

Yet another approach relies on the fact that circuit delays rise with temperature in a quantifiable way (see Eq. (17) below) [54]. A circuit comprising an odd number of inverters connected in a ring will therefore oscillate at a rate linked to the propagation delay of a signal around this ring. Measuring the frequency of oscillation can yield an estimate of the temperature [90]. We can, for instance, use a counter to count the number of oscillations in a prespecified time period: a conversion table then associates a temperature with each count. Studies in [90] suggest that the oscillator frequency is approximately linearly decreasing with temperature; over the range 0–125 °C, they observed the frequency drop by about 12%. Furthermore, a study of the impact of process variation on the accuracy of this sensor indicates that when feature sizes are assumed to be normally distributed with $3\sigma = 20\,\%$ of the nominal value (σ is the standard deviation), the impact on sensor accuracy is extremely small.

Performance counters are commonly found in high-end microprocessors for the purpose of monitoring the performance of the processor, and can provide an indirect means for estimating the temperature [48]. These programmable counters count certain architectural events (such as cache misses or branch mispredictions) as selected by the user (normally, the system administrator). The power consumption of a component C can be expressed as [39]:

$$\begin{aligned}\mathrm{Power}(C) = {} & \mathrm{AccessRate}(C) \times \mathrm{Scaling}(C) \times \mathrm{MaxPower}(C) \\ & + \mathrm{NonGatedClkPower}(C) \qquad (3)\end{aligned}$$

where AccessRate(C) denotes the access rate of that component, usually derived from a combination of several performance counters, while the other terms (a scaling constant, the maximum power of that component, and the non-gated clock power) are constants that can be obtained by experimentation on the chip or a power simulator. The total power is the power consumed by all these components plus the idle power (i.e. that consumed when there is no activity). Given the estimates of the power consumed by the individual blocks in the chip and a model of heat flow, one can obtain the time-dependent temperature distribution of the entire chip [27,48] as is described in the next section.

2.2. Thermal modeling

Simulation-based algorithms to estimate on-chip temperatures have been developed (e.g. [84]). The first step of the algorithm uses a power simulator to generate a power trace, consisting of the power consumption of the various subunits of the chip, taken at specified sampling instants. For simplicity, power consumption is assumed to be constant during each inter-sampling interval. The power consumed by each unit turns into heat, and in the second step of the algorithm the temperatures of the individual subunits are estimated. To this end, a heat flow model that is equivalent to an electrical RC circuit is used with the heat generation process in each subunit modeled as a current source. Standard circuit-theory techniques can then be used to obtain the temperature at each node (i.e. chip subunit). The resulting equations are linear and as a result, linear system theory techniques can be used to ensure a rapid solution.

Power simulators, such as Wattch [10], are well known and will not be discussed here. We present next the widely used *Hotspot* heat flow model [84].

2.2.1. Hotspot heatflow model

The *Hotspot* model is based on an analogy between heat and electricity. Heat flow can be treated in the same way as current flow; just as the amount of current flowing from one point to another depends on the voltage differential between those points and the resistance of the path connecting them, the amount of heat flowing between two points depends on the temperature differential between them and the thermal resistance of the heat-flow path. Just as capacitance measures the charge-carrying capacity of a device in terms of the volume of charge required to raise its voltage by one unit, the heat capacitance measures its heat-carrying capacity in terms of the amount of heat energy required to raise its temperature by one degree. We can therefore model heat flow by means of an analogous RC electrical circuit.

The thermal resistance of a path is defined as the amount of heat flow along the path that is required to create a temperature differential of 1 °C across the path. It is proportional to the length of the path, L, divided by its cross-sectional area, A, i.e. $R_{thermal} = L/(kA)$ where k is a constant of proportionality, called the *thermal conductivity*, that depends on the material of which the path is composed, typically expressed in units of Watts/Kelvin/Meter. Its values for silicon are shown in Fig. 1.

As can be seen, k is sensitive to temperature: as the temperature increases, the thermal conductivity drops rapidly. This contributes to the thermal runaway problem: as the chip heats up, its ability to conduct away the heat drops, which can contribute to a further heating up of the chip.

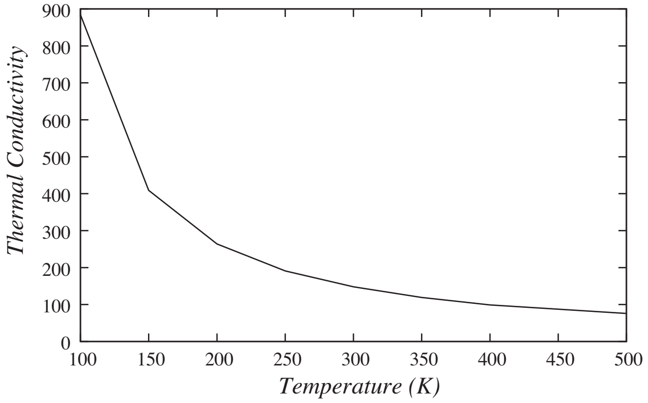

Fig. 1. Thermal conductivity of silicon (Watts/Kelvin/Meter).

Thermal capacitance is the amount of heat energy required to increase the temperature of the given subunit by one degree. The thermal capacitance of any quantity of material is proportional to its mass, M, i.e. $C_{thermal} = cM$, where c is a constant called the *specific heat capacity*. This is commonly expressed in units of Joules/Kelvin/Kg. The heat capacity of silicon tends to increase with temperature (see Fig. 2). Since the density of the material is known, we can also express the thermal capacitance as a function of the volume of the entity in question.

With this background, we now present the three heat flow models that are needed: one for lateral heat flow, and the other two for vertical heat flow. Heat is modeled as being generated at the center of each block (subunit) and flowing outward from there.

An example of the lateral model is shown in Fig. 3.

We have in this example four blocks, and the center of each block is connected to the interface of each of its neighbors by means of a resistance and a thermal capacitance. Ground represents the ambient environment.

The primary model of vertical heat flow is similar, and an example of it is shown in Fig. 4. The IC package has, atop it, a thermal package consisting of a *heat spreader* which conducts the heat away from the IC to a *heat sink*. It is obviously important that the heat spreader be highly conducting. Examples of materials used in heat spreaders include copper [2], diamond film [47], and (being studied for possible future use) graphene [85] (graphene has one of the highest thermal conductivities of all materials). Above the heat spreader is the heat sink.

The heat sink provides an *isothermal* layer, i.e. a layer with the same temperature in all lateral directions. A resistance from a block

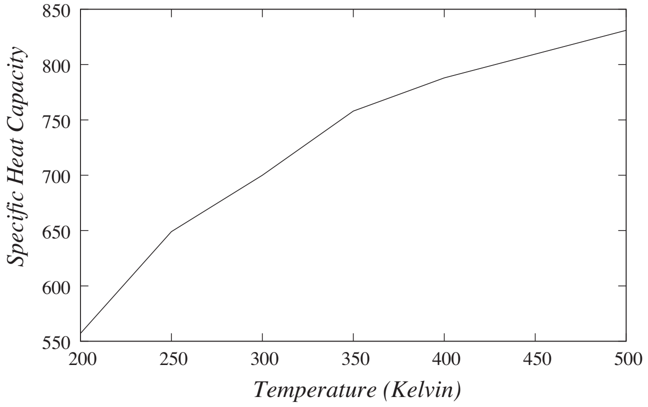

Fig. 2. Specific heat capacity of silicon (Joules/Kelvin/Kg).

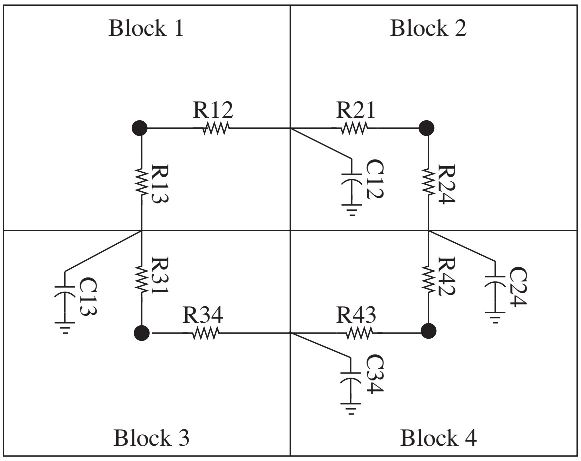

Fig. 3. Example of a lateral model of heat flow.

of the die (there are two lateral blocks in the example in Fig. 4) to the isothermal layer represents the heat flow from that block and the capacitance associated with the block represents its heat-holding capacity. A single resistance represents the heat flow from the isothermal layer to the heat sink and to the outside world.

The second model of vertical heat flow evaluates the heat flow through the on-chip interconnect and the ceramic substrate of the chip to the printed circuit board. It follows the same general rules as does the primary model: for details, see [32,33].

Given the parameter values of this RC circuit and the input from a power simulator indicating how much heat is generated at each of the nodes (representing the chip blocks), the differential equations governing the circuit can be solved and the temperature (which is the thermal analog of voltage) at each node obtained.

2.2.2. TILTS acceleration technique

If the system as modeled by *Hotspot* has a large number of nodes, it can be time consuming to solve the differential equations associated with the equivalent RC circuit. Considerable speedup can be achieved by exploiting the fact that the RC circuit results in linear differential equations [28]; these can be considered as forming the state equations of a linear system, which can then be solved using techniques from linear control theory.

An N-state-variable and M-input linear control system has a state variable vector $x(t)=(x_1(t), \cdots, x_N(t))^T$ and an input vector $u(t)=(u_1(t), \cdots, u_M(t))^T$ satisfying the standard linear system equation:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{4}$$

where A and B are $N \times N$ and $N \times M$ matrices, respectively. This equation has the solution

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau \tag{5}$$

Denote now by M the number of units within the processor that generate heat; their heat inputs to the system constitute $u(t)$. Denote by $x(t)$ the temperature vector; that is the state variable of our linear control system. Denote the thermal resistance between nodes i and j by r_{ij} and the thermal capacitance of node i by c_i. For notational convenience, define $r_{ii}=\infty$ and let $\tilde{u}(t) = (u_1(t), \cdots, u_M(t), 0, \cdots 0)^T$ be an N-sized vector ($M \leq N$ is always satisfied). Then, the *Hotspot* RC circuit equations reduce to the differential equation

$$c_i\dot{x}_i(t) = -\sum_{j=1}^{N} \frac{1}{r_{ij}}(x_i(t) - x_j(t)) + \tilde{u}_i(t) \tag{6}$$

This can be written in matrix form as

$$C\dot{x}(t) = Dx(t) + \tilde{u}(t) \tag{7}$$

where C is the diagonal matrix $[\xi_{ij}]$ with $\xi_{ii}=c_i$ and $\xi_{ij}=0$ for $i \neq j$. Since C is a non-singular matrix, we can rewrite the equation in standard linear form

$$\dot{x}(t) = C^{-1}Dx(t) + C^{-1}\tilde{u}(t) \tag{8}$$

$$\dot{x}(t) = Ax(t) + B\tilde{u}(t) \tag{9}$$

where $A=C^{-1}D$ and $B=C^{-1}$.

The power trace input to the thermal model provides temperature samples taken every Δt seconds. Over the period between two consecutive samples, the power consumption vector is assumed to be fixed, say at u. From the above expressions we can write:

$$x(\Delta t) = e^{A\Delta t}x(0) + \left[\int_0^{\Delta t} e^{A(\Delta t-\tau)}Bd\tau\right] \cdot u \tag{10}$$

Denoting $F=e^{A\Delta t}$ and $G = \int_0^{\Delta t} e^{A(\Delta t-\tau)}B$, we obtain the following difference equation:

$$x(\Delta t) = Fx(0) + Gu \tag{11}$$

Assuming that for the time interval of interest the thermal capacitances and resistances are approximately constant, the above linear system is time-invariant and the difference equation is preserved under time-translation:

$$x(n\Delta t) = Fx([n-1]\Delta t) + Gu(n-1) \tag{12}$$

where $u(n)$ is the input vector applied during the nth sampling interval. To simplify the notation we remove Δt and rewrite the difference equation as follows:

$$x(n) = Fx(n-1) + Gu(n-1) \tag{13}$$

Since F and G can be computed just once and stored, this leads to a fairly rapid solution approach as compared to solving the differential equations every time.

This approach provides additional savings if we only need the value of $x(n)$ over a coarser time granularity, say every $k\Delta t$ seconds. In such a case, we can use Eq. (13) to write:

$$\begin{aligned} x(n+k) &= Fx(n+k-1) + Gu(n+k-1) \\ &= F^2x(n+k-2) + FGu(n+k-2) + Gu(n+k-1) \\ &\vdots \\ &= \vdots \\ &= F^kx(n) + F^{k-1}Gu(n) + \cdots + FGu(n+k-2) + Gu(n+k-1) \end{aligned} \tag{14}$$

We can precompute F^i, F^iG, $i=1, \ldots, k$, and use them repeatedly.

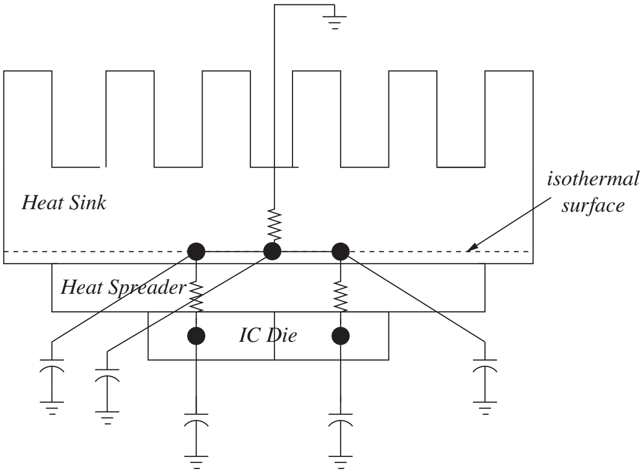

Fig. 4. Example of primary model of vertical heat flow.

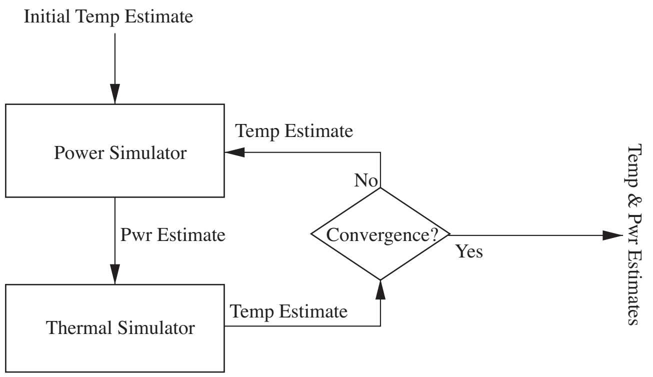

Fig. 5. Iterative power and temperature estimation.

2.2.3. *Iterative use of thermal models*

In the previous section, we assumed that a power simulator, such as Wattch, is used to generate a power trace that would be the input for the thermal simulator. However, this approach requires refinement if we are modeling a technology where leakage is a significant part of the total power consumption, since the leakage is considerably affected by the temperature.

This can be done by placing both power and thermal simulators into an iterative loop as shown in Fig. 5 [8].

The power simulator starts with some initial estimate of the temperature. It then computes the power consumption of each of the blocks and inputs them to the thermal simulator, which estimates their temperature. This value is fed back to the power simulator, which refines its power estimate based on the new temperature estimate. The computation iterates until convergence is attained.

For an alternative, non-iterative, approach that exploits the fact that the thermal time constant of the package (i.e. heat spreader and heat sink) is fairly long, see [68].

3. Temperature-aware chip design

Several design techniques, that either reduce the maximum temperature that the chip can experience or provide hardware support for run-time temperature management, have been developed. The first approach mainly focuses on the floorplanning phase of the physical design process. An example of the second approach is duplicating a block that is likely to become a hotspot. The second copy of the block will become active when the temperature of the first copy exceeds a predetermined threshold.

The conventional goals of floorplanning are minimization of the total chip area and minimization of the wiring delays between blocks that lie on a critical execution path. However, the floorplan of a chip also affects the maximum temperature that the chip blocks reach, because of the lateral heat transmission that occurs between adjacent blocks. Several researchers have made this observation (e.g. [11,18,29,37,74]) and most of them have presented techniques to either modify an existing floorplan or modify the floorplanning algorithm to include the maximum temperature as an additional objective.

To illustrate the impact of a floorplan on the temperature, the maximum temperature of the hottest block in an Alpha microprocessor chip has been studied in [29] for several different floorplans for that chip. The results have shown that the maximum temperature can be as high as 132 °C and as low as 95 °C. These are the estimated temperatures for the Integer Register File (*IntReg*) block during the execution of the *gcc* SPEC2000 benchmark.

However, the two floorplans that had these two extreme temperatures would not necessarily be considered as candidate floorplans in practice since the corresponding wiring delays were higher than the smallest possible value of these delays. The maximum temperature of the *IntReg* block (for the *gcc* benchmark) in the original floorplan of the Alpha chip (shown in Fig. 6) has been estimated to be 120 °C. Fig. 6 shows only the processor core (the L2 cache unit is not shown) and indicates the maximum temperatures of the individual blocks during the execution of the *gcc* benchmark.

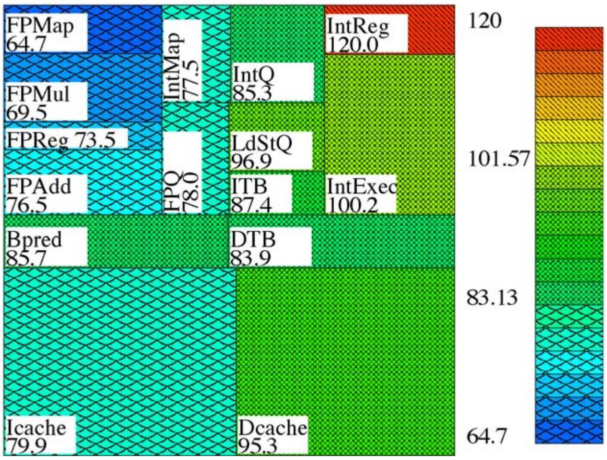

Fig. 6. The original Alpha (EV6) floorplan.

Furthermore, even if all possible floorplans of a certain chip, irrespective of the resulting wiring delay, are considered, the range of the maximum temperature would not necessarily be as high as 95–132 °C. A similar study of the Pentium Pro processor has shown [29] that the range of maximum temperatures for this chip is only between 94 °C and 111 °C. The main reason for the much smaller range is the more uniform distribution of the power densities of the individual blocks in the Pentium Pro processor.

Floorplanning algorithms, which ordinarily have the objectives of better performance (i.e. smallest weighted wire length where wires on a critical path are assigned higher weights) and lower total chip area, can be modified to also include reducing the maximum temperature of a block in the chip. Since floorplanning, in general, is an NP-complete problem, search algorithms such as simulated annealing [29,74] and genetic [37] algorithms have been used, requiring the estimation of the maximum temperature in every step, using for example, a tool such as *Hotspot*. Even if an acceleration scheme like *TILTS* is employed, the computational complexity of the floorplanning process greatly increases. To reduce this complexity, the calculation of the estimated temperature has been replaced in [29] by a simpler-to-compute proxy called Total Heat Diffusion. This proxy focuses only on the blocks with the highest power density and estimates the increase in their temperature due to the diffusion of heat from their neighboring blocks, calculated as $(d_i - d_j) \cdot shared\_length$ where d_i, d_j are the power densities of block i and j, respectively, and $shared\_length$ is the length of the boundary between the two blocks i and j.

Using this proxy within a simulated annealing procedure the modified floorplan shown in Fig. 7 was generated [29] with a maximum temperature (of the *IntReg* block for the *gcc* benchmark) of 99 °C. This floorplan was generated with the objectives of minimal wiring length and smallest maximum temperature. If the only goal of the floorplanning algorithm is minimal wiring length, the maximum temperature increases to 120 °C.

The reduction in the maximum temperature is sometimes achieved during the floorplanning process by increasing the distance between two high power-density blocks that happen to lie on a critical path of the pipeline, thus negatively impacting the performance. Therefore, the final floorplan generated when both

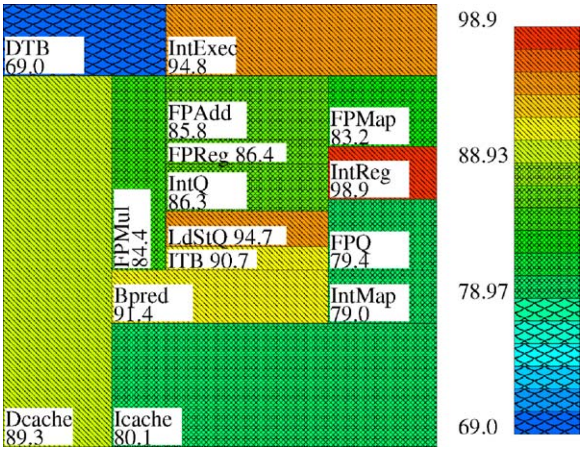

Fig. 7. A floorplan for the Alpha (EV6) processor with both short wire length and low maximum temperature.

wiring length and temperature are considered, represents a tradeoff between the maximum temperature and the performance. Some of these modifications that would increase the distance between two blocks can still be accepted after introducing dummy pipeline stages as described in [11].

Design techniques to reduce the maximum temperature on a chip should, however, not be limited to the floorplanning stage since floorplanning can only control the lateral heat transfer that is expected to be smaller than the vertical heat transfer (to the heat sink) [30]. Skadron et al. proposed Migrating Computation, whereby a hot unit is duplicated and the backup unit is used when the original unit becomes too hot [81]. To achieve a large reduction in the expected maximum temperature, the backup unit should be located away from the original one. Clearly, frequent migrations between the two copies of the hot block can better control the temperature. On the other hand, performing such migration at a high frequency can greatly increase the performance penalty due to too frequent copy operations [30]. Performing the migration at the level of microseconds provides an acceptable tradeoff between the two objectives.

Based on the observation (made in several studies) that the Integer Register File unit (*IntReg*) is the hottest block in a microprocessor for many benchmark applications, a new register file access policy has been proposed in [4]. According to this policy, registers are assigned (by the compiler) using a stride that is not equal to 1 to reduce the probability that two adjacent registers will be accessed frequently, thus resulting in mutual heating.

Another design technique to reduce the maximum temperature of a hot block has been discussed in [19]. The authors have shown that increasing the silicon area allocated to hot blocks can reduce the maximum temperature in multi-core and simultaneous multithreading processors by 5–11 °C. Unfortunately, the granularity of spreading the extra silicon area within the block must be very small in order to achieve the full benefits of this approach, thus greatly increasing the resulting design complexity and area overhead.

4. Runtime thermal management

4.1. Motivation

Runtime thermal management techniques are motivated by the following considerations:

- Runtime techniques offer control at a fine level of granularity. Computational workloads vary in their demands on the processor with time; for example, one phase of a program may involve very heavy use of the floating-point unit while another may stress the integer register file. The region of the processor that is thermally stressed will therefore vary with time and thermal management has to keep up with that. The thermal profile of the workload may also be a very sensitive function of the input parameters: these are very difficult to predict ahead of the execution.
- We can be much more aggressive with runtime thermal management than with a purely static approach. In the latter, we have to estimate the worst-case situation with respect to a given task allocation and handle the processor accordingly. Such an estimate must necessarily have a substantial safety margin built into it. Because a runtime technique reacts to the actual thermal situation at any given moment, tasks can be scheduled aggressively with the knowledge that when temperatures approach their safe bounds, runtime control is available as a backstop.
- Modern high-end processors have a significant number of "knobs" for the control of heat generation: we can control the processor at different levels of spatial granularity. For example, fetch throttling reduces the overall activity level of the chip by reducing the rate at which instructions are fetched. Fetch throttling can be carried out by the hardware or directed by the compiler (see [88] and the references therein). By contrast, clock gating an individual unit produces a rather focused cooling of that unit (or a set of units) [41].
- Both reactive and proactive approaches are possible. In a reactive approach, the system is reacting to the system known to be approaching its temperature bounds. In the proactive approach, the system can learn about the thermal characteristics of its workload and manage computational resources in such a way that overheating is avoided while keeping the performance impact as low as practical.

All runtime approaches have to keep in mind the underlying thermal time-constant of the hardware. This gives an indication of how fast the temperature can change as a function of the heat dissipation. Typical time constants quoted in the literature range from hundreds of microseconds [79] to many milliseconds [54]. Considerably less time is needed to raise the temperature of a processor or even overheat it if a *power virus* code (e.g. [61,87]) is executing.

The amount of information available about the workload affects the range of options available to the system. If no information is available, the only recourse is to identify when a temperature threshold has been crossed and then to slow down or stop the processor. If we have at least some profiling information about a task, this can be factored into the decision-making process. For example, some tasks may be intrinsically "cold" and others "hot" [69]. Alternatively, a task may go through different phases of thermal behavior. Such information may, in rare instances, be collected in advance, or more commonly, be learned by the system itself during operation [91,93].

In the rest of this section, we consider various ways in which to dynamically manage resources to avoid overheating.

4.2. Dynamic voltage scaling

Dynamic voltage scaling is one of the first run-time power and temperature management techniques to be implemented (e.g. [64]) and is still the one that is most commonly used. In this section, we describe the relationship between the voltage level and the corresponding heat dissipation, and indicate the performance reduction that must be expected when the voltage is lowered.

The energy dissipated by a processor is a strong function of its supply voltage. In CMOS devices, the dynamic power consumption, P_d, is given by

$$P_d(v) = C_L N_{sw} v^2 f \tag{15}$$

where v is the supply voltage, C_L is the circuit load capacitance, N_{sw} the number of switches per clock cycle, and f the clock frequency [44].

To obtain the total power consumption, we need to add to the dynamic power the leakage power, P_ℓ, that can be approximated by [54]

$$P_\ell(v) = AT^2 \quad \exp\left(\frac{(av+b)}{T}\right) + B \quad \exp(cv+d) \tag{16}$$

where A, B, a, b, c, d are constants and T is the absolute temperature.

The circuit delay δ increases as the voltage drops and as the temperature increases, according to the expression

$$\delta(v) = \frac{C_L v T^{\mu}(v)}{K(v - V_T)^{\alpha}} \tag{17}$$

where V_T is the threshold voltage, $T(v)$ the temperature at voltage v, K is a constant, and α and μ are constants (a typical value for each is approximately 1.2) [54]. The steady-state temperature, $T(v)$, is that at which the rate of heat generation matches the rate of heat dissipation. If the clocking is at the highest rate possible, when the supply voltage drops so too must the clock rate, by the above equation. Suppose v_h is the maximum supply voltage, we can define a slowdown factor at supply voltage v as follows:

$$\text{slow}(v) = \frac{v}{v_h}\left(\frac{v_h - v_T}{v - v_T}\right)^{\alpha}\left(\frac{T(v)}{T(v_h)}\right)^{\mu} \tag{18}$$

From these expressions, it is clear that the power consumption is a strongly nonlinear function of the supply voltage. At the cost of a roughly linear increase in the execution time, heat dissipation can be dramatically lowered by reducing the supply voltage. The aggressiveness with which the control voltage is lowered depends on the tradeoff between reducing thermal stress and the desire for performance. A small performance reduction can give rise to a substantial reduction in heat dissipation. However, we should keep in mind that the potential heat reduction depends on the range of supply voltages that can be applied: as the upper voltage limit has dropped from 5 V a few years ago to just over 1 V, the scope for savings is diminished.

Experiments have shown that two levels of voltage are generally sufficient for effective thermal management [78]. The time penalty for switching voltage levels is typically several microseconds [78].

4.3. *Thermal-aware task assignment and scheduling*

The operating system controls the scheduling of tasks, and in multicore systems, it also assign tasks to individual cores. Both can be exploited to prevent excessive processor heating [15].

4.3.1. *Heat balancing*

Heat balancing is analogous to load balancing in distributed systems [15]. As the term implies, it uses the thermal profiles of tasks to assign them to individual cores (and to migrate them during execution) in such a way that no one core is disproportionately loaded with “hot” tasks.

One implementation of this approach is to use information on *hotspot imbalance* [20]. For any given core, its hotspot imbalance is the difference in temperature between that core's hottest and second-hottest subsystems. Cores are considered for thread migration in order of their hotspot imbalance; the operating system looks for a thread that is likely to reduce the temperature of its hottest subsystem by the greatest amount.

Another approach, which uses a simple on–off approach to thermal management, is to classify a thread based on how quickly it took the core temperature to reach a predefined threshold [58]. Call this time τ_i for thread *i*. If $\tau_i < 20t_{off}$, thread *i* is considered hot, where t_{off} is the duration for which thread *i*'s core had to be turned off to cool down sufficiently. τ_i is used as a measure of the intrinsic hotness of thread *i*. The hottest threads can then be migrated to the coldest cores. To prevent migrations from happening too often, their frequency can be limited to, say no more than one per millisecond.

4.3.2. *Heat unbalancing*

It might seem counter-intuitive, but it is possible to argue for a policy (up to a point) of deliberately unbalancing the thermal load associated with the task assignments.

Heat balancing can potentially cause frequent *thermal cycling*, i.e. the repeated heating up and cooling down of subunits of the core. Such cycling can itself reduce the lifetime of the chip. To avoid this, while still keeping the chips from overheating, a more complicated approach has been suggested [53]. Here, based on their thermal profile, tasks are classified according to whether they are intense enough to trigger overheating (these are called hot-hazard tasks) and the rest. Hot-hazard jobs are assigned the coldest processors so as to minimize the need for throttling or voltage scaling. The other jobs are assigned in such a way as to deliberately unbalance the heat production: hotter (but not hot-hazard) jobs are assigned to hot processors. The aim is to reduce thermal cycling with respect to the non-hazard tasks while reducing the performance loss that would be incurred by triggering excessive thermal management measures with respect to the hot-hazard tasks.

A second argument for heat unbalancing is as follows: when the subsystem of a processor crosses its safety threshold, the *entire* processor often needs to be slowed down or even stopped (since the overheating subsystem presumably is a necessary part of the execution chain). In such a situation, it is all the same whether that slowdown occurs because the temperature threshold was breached by one subunit or several. Hence, it pays to have a group of threads assigned to a processor with, preferably, complementary demands on the processor, so that when it has to be slowed down or stopped, multiple subunits are at, or close to, their thermal limits. This is the *Heat-and-Run* approach [23].

4.3.3. *Reducing execution rate of hot tasks*

When a task (or thread) causes excessive heating, it can be scheduled less often by the operating system. That is, a certain fraction of its time-slices can be given up to a cooler task [15]. A related approach is to always select, when the temperature exceeds a certain limit, a task that causes minimum heating to the hottest block on the chip [46]. Yet another approach is core idling. This can be accomplished by inserting nops [15] or by fetch throttling or voltage scaling. Another approach is chip-wide clock gating: when the clock is frozen, the system's state is preserved but no dynamic activity takes place [20]. Such clock gating does slow down execution, however. In addition to such chip-wide gating, one can be more focused, and identify circuit elements that are not being used and clock-gate them. A study of such clock-gating in a superscalar pipeline is provided in [49].

4.3.4. *Adding a predictive component*

In our previous discussion of task-migration approaches, tasks were classified according to the temperature they produced. While it is possible to make this classification based on a prior thermal profiling of tasks or a simple extrapolation, more sophisticated techniques can be used to predict the thermal contributions of a task, given its recent history.

Such a prediction over at least the short term is aided by the fact that the thermal time-constants of most chips are fairly long (recall the 10–200 ms figure quoted earlier), i.e. temperatures can be expected to change only slowly. Thus, the temperatures can be regarded as the output of a frequency-limited process and as a result, extrapolation techniques from signal theory can be used [6].

Such techniques allow us to write the predicted signal value (temperature, in our case) as the weighted sum of prior recent signal values. More precisely, if $x(t)$ is the temperature at some location on the chip at time t, then we can write a prediction

$$x(t) \approx \sum_{n=1}^{N} a_n x(t_n) \tag{19}$$

where $t_i = t - i\tau_s$ $(i = 1, 2, \cdots N)$ are the N temperature sample points (for a suitable N) preceding time t. we assume here that these sample points are taken at uniform intervals of τ_s, although that is not necessary. If most of the signal "energy" is contained within a frequency range Ω, it can be shown that the coefficients of the best estimate, a_n, follow the equation:

$$\sum_{n=1}^{N} a_n \mathrm{sinc}(2\Omega(t_j - t_N)) = \mathrm{sinc}(2\Omega(t - t_j)), \quad j = 1, \cdots, N \tag{20}$$

where $sinc(x) = sin(x)/x$. Ω can be obtained through the thermal time-constant, τ_c: in [6], it is argued that, to a good approximation,

$$Temp(\omega) = \frac{Temp(0)}{\sqrt{1 + (\omega\tau_c)^2}} \tag{21}$$

where $Temp(\omega)$ is the level of the temperature signal at frequency ω. An appropriate level for the sampling interval, τ_s, can be obtained by invoking the Nyquist sampling theorem yielding $\tau_s < 1/(2\Omega)$.

Calculation of the a_i values depends on the thermal time-constant and can be done offline and stored in a lookup table. Temperature prediction at the next sampling point then becomes a fairly lightweight computation.

Other predictive approaches can be found in [17,45,91] which use regression techniques. These temperature predictions are generally accurate to within a few percent. They allow the operating system to act proactively, migrating tasks which are predicted to become very hot to cooler cores.

Table 1 summarizes the key runtime approaches to thermal management.

4.4. Real-time systems

Tasks in a real-time system have deadlines and missing these deadlines could either lead to an unacceptable outcome (for hard real-time systems like engine controllers) or to degraded performance (for soft real-time systems, e.g. streaming video). Such systems are generally provisioned with sufficient computational power to meet the deadlines of their critical tasks under worst-case execution time conditions. However, in most cases, tasks do not run to their worst-case estimates. As a result, power consumption – and heat generation – can be lowered by taking advantage of the extra time that becomes available and reducing the voltage and frequency of the processor. Voltage scaling for reducing the total energy consumption has long been an active research area in real-time systems [44,65,86]. Temperature constraints constitute an additional dimension in the scheduling of such tasks; they limit the maximum voltage that can be used for a given task. Since this maximum voltage can be limited by the temperature of the various blocks within the processor when the task starts executing, thermal limits create an interaction between tasks which is not accounted for in traditional real-time scheduling.

Table 1
Runtime techniques.

Voltage scaling	Change voltage levels to adjust power and energy consumption. Clock rates are reduced to match the increased circuit delay that results.
Heat balancing	Spreads the thermal load among multiple cores to approximately even out their temperatures.
Heat unbalancing	Reduce thermal cycling effects: accept significant temperature differentials between the cores as long as specified temperature levels are not breached.
Throttling	Reduce the rate at which heat is generated by reducing instruction fetch rate and similar parameters.

Some work has been reported in thermal-aware scheduling in real-time systems, although the field is still in its infancy. In [67], the authors prove that using the lowest speed that guarantees that all deadlines are met minimizes the maximal temperature. In [22], a two-loop feedback control system is used to manage *soft* real-time systems. The outer (periodic) loop uses temperature information to set the processor utilization goal for the inner loop. The inner loop adjusts the rate at which the tasks execute in order to meet the utilization goal. The assumption is that the application task periods can be varied within a given range. If, for example, the temperature rises too high, the task iteration rate can be throttled back, within limits.

In [89], the authors use reactive speed-scaling. When there are tasks waiting to run and the temperature is below a predefined threshold, the processor is run at full speed. When the threshold is reached, the processor speed is reduced to that which ensures that thermal equilibrium is maintained (i.e. the temperature does not rise). Note that the temperature is regarded here as a scalar, i.e. one measurement which applies across the chip. Extending this algorithm to the more complex – and realistic – case where each block has its own temperature should be quite simple.

In [92], the focus in on scheduling to reduce the amount of leakage. As we have seen, leakage current rises strongly with temperature. Hence, it is better to run a processor when it is cold and send it to sleep when it becomes hot (so long as deadlines are not missed). A heuristic is used to obtain, for single-task workloads, the intervals over which the processor is to be active. The time between the release of the task and its deadline is divided into small subintervals; the heuristic specifies whether the processor is to be awake or asleep in each such subinterval. Let T_s be the steady-state temperature if the processor is running at the specified voltage level (the assumption is that this temperature is below the safety threshold). Let T_a be the ambient temperature (we assume in this description that the steady-state temperature when the processor is sleeping is very close to T_a). The heuristic calculates the ratio, r_i, of the remaining work on the task to the available time in each subinterval, i. Whether or not the processor sleeps during subinterval i depends on the value of r_i and how the current temperature relates to the temperature range $[T_a, T_s]$. If the current temperature is T, then if $r_i < (T - T_a)/(T_s - T)$, the processor sleeps during subinterval i, otherwise it executes the task. Note that this heuristic also assumes the temperature is characterized by one number rather than by a vector of block temperatures.

Another heuristic for minimizing the maximum temperature on a chip has been suggested in [12]. They do offline assignment (to cores) and scheduling of periodic real-time tasks. Since the tasks are periodic, one can compute the schedule over a hyperperiod, i.e. the least common multiple of all the task periods; the schedule will repeat every hyperperiod. The heuristic in [12] consists of doing a binary search to minimize the maximum block temperature. A simple list scheduling heuristic accepts as input the range of maximum block temperatures that are allowed and either returns an assignment/schedule that falls within this range or else reports failure. This heuristic can be used in a binary search algorithm (with a specified maximum number of iterations) to find a task assignment and schedule which approximately minimize the maximum temperature.

5. Discussion

Thermal dissipation is today a significant – and costly – problem in processor design and operation. Thermal runaway effects have the potential to destroy a processor; even at temperatures lower than those causing thermal runaway, high temperatures can have a debilitating effect, reducing the processor's lifespan. To ameliorate the thermal stress on a processor while preserving high performance requires measures to be taken during both the processor design phase and its operation. Implementing these measures requires effective means of predicting the temperature during the design stage and of measuring it (directly or indirectly) during operation.

In this paper, we have outlined techniques to do so. We have also discussed design techniques to facilitate more effective heat dissipation. Since it is no longer feasible, in many instances, to provide a cooling infrastructure for the case in which all processor subunits are generating worst-case amounts of heat, we have described mechanisms to detect excessive heating during processor operation and slow down or stop subunits to allow them to cool. Such slowdown can be done at both the hardware and operating system levels. It can be system-wide, core-wide, or focused on specific overheating subunits. Additionally, the operating system can reassign threads to cores based on their thermal characteristics.

Variants of most of the techniques that we have presented in this paper have been incorporated in current high-end microprocessors, especially the dual- and quad-core ones. Due to the complexity of some of these techniques and the need for a very short response time, some processors include a custom on-die microcontroller that is dedicated to the task of temperature and power management (e.g. [56,71]).

As transistor densities increase, we can expect thermal constraints to remain significant factors in processor design and operation. The thermal-aware techniques surveyed here and new ones that are yet to be developed, will likely play a major role in future systems.

Acknowledgment

This work has been supported in part by the National Science Foundation under grant CNS-0931035.

References

[1] http://www.dibeneditto.com/resources/20011105/.

[2] N. Amin, V. Lim, F.C. Seng, R. Razid, I. Ahmad, A practical investigation on nickel plated copper heat spreader with different catalytic activation processes for flip-chip ball grid array packages, Microelectronics Reliability 49 (2009) 537–543.

[4] J.L. Ayala, A. Apavatjrut, D. Atienza, M. Lopez-Vallejo, Exploring temperature-aware design of memory architectures in VLIW systems, in: Workshop on Innovative Architecture for Future generation Processors and Systems, 2007, pp. 81–88.

[6] R. Ayoub, T. Rosing, Predict and act: dynamic thermal management for multi-core processors, in: International Symposium on Low Power Electronics and Design (ISLPED), 2009.

[8] M. Bao, A. Andrei, P. Eles, Z. Peng, Online thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration, in: Design Automation Conference (DAC), 2009, pp. 490–495.

[10] D. Brooks, V. Tiwari, M. Martonosi, Wattch: a framework for architectural-level power analysis and optimizations, in: International Symposium on Computer Architecture (ISCA), 2000, pp. 83–94.

[11] A. Chakravorty, A. Ranjan, R. Balasubramonian, Re-visiting the performance impact of microarchitectural floorplanning, in: Third Workshop on Temperature Aware Computer Systems (TACS), 2006.

[12] T. Chantem, R.P. Dick, X.S. Hu, Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs, in: Design, Test and Automation in Europe (DATE), 2008, pp. 288–293.

[15] J. Choi, C.-Y. Cher, H. Franke, H. Hamann, A. Weger, P. Bose, Thermal-aware task scheduling at the system software level, in: International Symposium on Low-Power Electronics and Design (ISLPED), 2007, pp. 213–218.

[17] A. Coskun, T. Rosing, K. Gross, Proactive temperature management in MPSOC, in: International Symposium on Low Power Electronics and Design (ISLPED), 2008, pp. 165–170.

[18] D. Cuesta, J. Ayala, J. Hidalgo, M. Poncino, A. Acquaviva, E. Macii, Thermal-aware floorplanning exploration for 3D multi-core architectures, in: 20th Great Lakes Symposium on VLSI, May, 2010.

[19] J. Donald, M. Martonosi, Temperature-aware design issues for SMT and CMP architectures, in: 5th Workshop on Complexity-Effective Design, June, 2004.

[20] J. Donald, M. Martonosi, Techniques for multicore thermal management: classification and new exploration, in: International Symposium on Computer Architecture, 2006.

[22] Y. Fu, N. Kottenstette, Y. Chen, C. Lu, X.D. Koutsoukos, H. Wang, Feedback thermal control for real-time systems, in: IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010, pp. 111–120.

[23] M. Gomaa, M.D. Powell, T.N. Vijaykumar, Heat-and-run: leveraging SMT and CMP to manage power density through the operating system, in: Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2004.

[24] S.H. Gunther, F. Binns, D.M. Carmean, J.C. Hall, Managing the impact of increasing microprocessor power consumption, Intel Technology Journal March (Q1) (2001) 1–9.

[25] H.F. Hamann, A. Weger, J.A. Lacey, Z. Hu, P. Bose, E. Cohen, J. Wakil, Hotspot-limited microprocessors: direct temperature and power distribution measurements, IEEE Journal of Solid-State Circuits 42 (January (1)) (2007) 56–65.

[27] Y. Han, I. Koren, C.M. Krishna, Temptor: a lightweight runtime temperature monitoring tool using performance counters, in: Third Workshop on Temperature-Aware Computer Systems (TACS-3), June, 2006.

[28] Y. Han, I. Koren, C.M. Krishna, "TILTS: a fast architectural-level transient thermal simulation method, Journal of Low-Power Electronics 3 (April (1)) (2007) 13–21.

[29] Y. Han, I. Koren, Simulated annealing based temperature aware floorplanning, Journal of Low-Power Electronics 3 (September (2)) (2007) 141–155.

[30] S. Heo, K. Barr, K. Asanovic, Reducing power density through activity migration, in: International Symposium on Low power Electronics and Design, August, 2003.

[31] http://www.hpl.hp.com/research/smart_cooling.

[32] W. Huang, M. Stan, K. Skadron, K. Sankaranarayan, S. Ghosh, S. Velusamy, Compact thermal modeling for temperature-aware design, in: Design Automation Conference (DAC), 2004, pp. 878–883.

[33] W. Huang, M. Stan, K. Skadron, K. Sankaranarayan, S. Ghosh, S. Velusamy, Compact thermal modeling for temperature-aware design, in: Technical Report CS-2004-13, University of Virginia, Department of Computer Science, 2004.

[37] W.-L. Hung, C. Addo-Quaye, T. Theocharides, Y. Xie, N. Vijaykrishnan, M.J. Irwin, Thermal-aware floorplanning using genetic algorithms, in: International Symposium on Quality Electronic Design (ISQED), March, 2005, pp. 634–639.

[39] C. Isci, M. Martonosi, Runtime power monitoring in high-end processors: methodology and empirical data, in: International Symposium on Microarchitecture (MICRO-36), December, 2003.

[40] P. Ituero, J.L. Ayala, M. Lopez-Vallejo, Leakage-based on-chip thermal sensor for CMOS technology, in: IEEE International Symposium on Circuits and Systems (ISCAS), 2007, pp. 3327–3330.

[41] S. Kaxiras, M. Martonosi, Computer Architecture Techniques for Power Efficiency, Morgan and Claypool, 2008.

[44] C.M. Krishna, Y.-H. Lee, Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems, IEEE Transactions on Computers 52 (December (12)) (2003) 1586–1593.

[45] A. Kumar, L. Shang, L.-S. Peh, N.K. Jha, HybDTM: a coordinated hardware-software approach for dynamic thermal management, in: Design Automation Conference (DAC), 2006, pp. 548–553.

[46] E. Kursun, C.-Y. Cher, A. Buyuktosunoglu, P. Bose, Investigating the effects of task scheduling on thermal behavior, in: Third Workshop on Temperature Aware Computer Systems (TACS), 2006.

[47] C. Lee, J. Ward, R. Lin, E. Schlecht, G. Chattopadhyay, J. Gill, B. Thomas, A. Maestrini, I. Mehdi, P. Siegel, Diamond heat spreaders for submillimeter-wave gaas Schottky diode frequency multipliers, in: International Symposium on Space Terahertz Technology, 2009, pp. 43–46.

[48] K.-J. Lee, K. Skadron, Using performance counters for runtime temperature sensing in high-performance processors, in: International Parallel and Distributed Processing Symposium (IPDPS), April, 2005.

[49] H. Li, S. Bhunia, Y. Chen, T.N. Vijaykumar, K. Roy, Deterministic clock gating for microprocessor power reduction, International Symposium on High Performance Computer Architecture (HPCA) (2003) 113–122.

[53] L. Li, X. Zhou, J. Yang, V. Puchkarev, ThresHot: an aggressive task scheduling approach in CMP thermal design, in: Workshop on Unique Chips and Systems, 2009.

[54] W. Liao, L. He, K.M. Lepak, Temperature and supply voltage aware performance and power modeling at microarchitecture level, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24 (July (7)) (2005) 1042–1053.

[56] R. McGowen, C. Poirier, C. Bostak, J. Ignowski, M. Millican, W. Parks, S. Naffziger, Power and temperature control on a 90-nm Itanium family processor, IEEE Journal of Solid-State Circuits 41 (January) (2006) 229–237.

[57] G.C.M. Meijer, G. Wang, F. Fruett, Temperature sensors and voltage references implemented in CMOS technology, IEEE Sensors Journal 1 (October (3)) (2001) 225–234.

[58] P. Michaud, A. Seznec, D. Fetis, Y. Sazeides, T. Constantinou, A study of thread migration in temperature-constrained multicores, ACM Transactions on Architecture and Code Optimization 4 (June (2)) (2007) (Article 9).

[61] K. Najeeb, V.R. Konda, S.S. Hari, V. Kamakoti, V.M. Vedula, Power virus generation using behavioral models of circuits, in: 25th IEEE VLSI Test Symmposium, 2007, pp. 35–42.

[64] T Pering, T. Burd, R.W. Brodersen, The simulation and evaluation of dynamic voltage scaling algorithms, in: Int. Symp. on Low Power Electronics Design (ISPLED), 1998.

[65] P. Pillai, K.G. Shin, Real-time dynamic voltage scaling for low-power embedded operating systems, Operating Systems Review 35 (October (5)) (2001) 89–102.

[67] G. Quan, S. Ren, Leakage-aware real-time scheduling for maximal temperature minimization, ACM SIGBED Review 7 (January (1)) (2010) (Article No. 3).

[68] R. Rao, S. Vridula, Performance optimal processor throttling under thermal constraints, in: International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2007, pp. 257–266.

[69] E. Rohou, M.D. Smith, Dynamically managing processor temperature and power, in: 2nd Workshop on Feedback-Directed Optimization, 1999.

[71] E. Rotem, A. Cohen, J. Hermerding, H. Cain, Temperature measurement in the Intel Core Duo Processor, in: 12th International Workshop on Thermal Investigations of ICs (THERMINIC), September, 2006, pp. 23–27.

[74] K. Sankaranarayanan, S. Velusamy, M. Stan, K. Skadron, A case for thermal-aware floorplanning at the microarchitectural level, Journal of Instruction-Level Parallelism (7) (2005) 8–16.

[75] H. Sanchez, R. Philip, J. Alvarez, G. Gerosa, A CMOS temperature sensor for powerPC RISC microprocessors, in: Symposium on VLSI Circuits, 1997, pp. 13–14.

[78] K. Skadron, Hybrid architectural dynamic thermal management, in: Proc. DesignAutomation and Test in Europe Conference (DATE), 2004.

[79] K. Skadron, T. Abdelzahar, M.R. Stan, Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management, in: International Symposium on High Performance Computer Architecture, 2002.

[81] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, D. Tarjan, Temperature-aware microarchitecture, in: 30th Annual International Symposium on Computer Architecture (ISCA), 2003, pp. 2–13.

[84] M.R. Stan, K. Skadron, M. Barcella, W. Huang, K. Sankaranarayanan, S. Velusamy, HotSpot: a dynamic compact thermal model at the processor-architecture level, Microelectronics Journal 34 (2003) 1153–1165.

[85] S. Subrina, D. Kotchetkov, A.A. Balandin, Thermal management with graphene lateral heat spreaders: a feasibility study, EEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm) (2010) 1–5.

[86] O.S. Unsal, I. Koren, System-level power-aware design techniques in real-time systems, Proceedings of the IEEE 91 (July (7)) (2003) 1055–1069.

[87] R. Viswanath, V. Wakharkar, A. Watwe, V. Lebonheur, Thermal performance challenges from silicon to systems, Intel Technology Journal Q3 (2000) 1–16.

[88] H. Wang, Y. Guo, I. Koren, C.M. Krishna, Compiler-based adaptive fetch throttling for energy-efficiency, in: IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2006, pp. 112–119.

[89] S. Wang, R. Bettati, Reactive speed control in temperature-constrained real-time systems, in: Euromicro Conference on Real-Time Systems (ECRTS), 2006.

[90] Y.-W. Yang, K.S.-M. Li, Temperature-aware dynamic frequency and voltage scaling for reliability and yield enhancement, in: Asia and South Pacific Design Automation Conference, 2009, pp. 49–54.

[91] I. Yeo, C.C. Liu, E.J. Kim, Predictive dynamic thermal management for multicore systems, in: Design Automation Conference (DAC), 2008, p. pp 739.

[92] L. Yuan, S. Leventhal, G. Qu, Temperature-aware leakage minimization techniques for real-time systems, in: IEEE/ACM International Conference on Computer Aided Design (ICCAD), 2006, pp. 761–764.

[93] F. Zanini, D. Atienza, L. Benini, G. DeMicheli, Multicore thermal management with model predictive control, in: European Conference on Circuit Theory and Design (ECCTD), 2009, pp. 711–714.

Further reading

[3] D. Atienza, G. De Micheli, L. Benini, J.L. Ayala, P.G. Del Valle, M. DeBole, V. Narayanan, Reliability-aware design for nanometer-scale devices, in: 2008 Conference on Asia and South Pacific design automation, January, 2008.

[5] J.L. Ayala, D. Atienza, P. Brisk, Thermal-aware data flow analysis, in: 46th Annual Design Automation Conference, July, 2009.

[7] N. Bansal, T. Kimbrel, K. Pruhs, Speed scaling to manage energy and temperature, Journal of the ACM 54 (March (1)) (2007) (Article 3).

[9] L. Benini, G.D. Micheli, Dynamic Power Management: Design Techniques and CAD Tools, Kluwer, 1998.

[13] P. Chaparro, J. Gonzalez, A. Gonzalez, Thermal-aware Clustered Microarchitectures, in: International Conference on Computer Design (ICCD), 2004, pp. 48–53.

[14] P. Chaparro, G. Magklis, J. Gonzalez, A. Gonzalez, Distributing the frontend for temperature reduction, in: 11th International Symposium on High-Performance Computer Architecture, 2005, pp. 61–70.

[16] S. Chung, K. Skadron, Using on-chip event counters for high-resolution real-time temperature measurements, in: IEEE/ASME Tenth Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITHERM), June, 2006, pp. 114–120.

[21] www.efunda.com.

[26] Y. Han, I. Koren, C.M. Krishna, TILTS: a fast architectural-level transient thermal simulation method, Journal of Low-Power Electronics 3 (2007) 1–9.

[34] W. Huang, M.R. Stan, K. Sankaranarayanan, R.J. Ribando, K. Skadron, Many-core design from a thermal perspective, in: 41st Annual Conference on Design Automation (DAC), 2008.

[35] W. Huang, K. Skadron, S. Gurumurthi, R.J. Ribando, M.R. Stan, Differentiating the roles of ir measurement and simulation for power and temperature-aware design, in: 2009 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), April, 2009, pp. 1–10.

[36] W. Huang, K. Skadron, S. Gurumurthi, R.J. Ribando, M.R. Stan, Exploring the thermal impact on manycore processor performance, in: IEEE Semiconductor Thermal Measurement, Modeling, and Management Symposium (Semi-Therm 26), February, 2010.

[38] W. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir, Thermal-aware task allocation and scheduling for embedded systems, in: Design, Automation and Test in Europe (DATE), 2005, pp. 898–899.

[42] A. Keshavarzi, Power-aware architectural synthesis, in: Wai-Kai Chen (Ed.), The VLSI Handbook, CRC Press, 2006.

[43] M.M. Khellah, M.E. Elmasry, Power minimization of high-performance submicron CMOS circuits using a Dual-V_{dd} Dual-V_{th} (DVDT) approach, in: ACM International Symposium on Lower-Power Electronics and Design (ISLPED), 1998, pp. 106–108.

[50] H. Li, P. Liu, Z. Qi, L. Jin, W. Wu, S. Tan, J. Yang, Efficient thermal simulation for run-time temperature tracking and management, in: International Conference on Computer Design (ICCD), October, 2005, pp. 130–136.

[51] Y. Li, K. Skadron, Z. Hu, D. Brooks, Evaluating the thermal efficiency of SMT and CMP architectures, in: IBM T. J. Watson Conference on Interaction between Architecture, Circuits, and Compilers, October, 2004.

[52] Y. Li, K. Skadron, D. Brooks, Z. Hu, Performance energy, and thermal considerations for SMT and CMP architectures, in: 11th International Symposium on High-Performance Computer Architecture, HPCA-11, February, 2005, pp. 71–82.

[55] P. Liu, Z. Qi, H. Li, L. Jin, W. Wu, S. Tan, J. Yang, Fast thermal simulation for architecture-level dynamic thermal management, in: International Conference on Computer-Aided Design (ICCAD), November, 2005, pp. 639–644.

[59] F. Mulas, D. Atienza, A. Acquaviva, S. Carta, L. Benini, G. De Micheli, Thermal balancing policy for multiprocessor stream computing platforms, EEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 28 (December) (2009) 1870–1882.

[60] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, G. De Micheli, Temperature-aware processor frequency assignment for MPSoCs using convex optimization, in: 5th IEEE/ACM International Conference on Hardware/Software Codesign And System Synthesis, 2007, pp. 111–116.

[62] A. Naveh, E. Rotem, et al., Power and thermal management in the Intel Core Duo, Intel Technology Journal 10 (2006).

[63] G. Paci, P. Marchal, F. Poletti, L. Benini, Exploring Temperature-aware design in low-power MPSoCs, in: Conference on Design, automation and test in Europe, March, 2006.

[66] M.D. Powell, M. Gomaa, T.N. Vijaykumar, Heat-and-run: leveraging SMT and CMP to manage power density through the operating system, in: 11th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems, October, 2004.

[70] E. Rotem, A. Naveh, et al., Analysis of thermal monitor features of the Intel Pentium M Processor, in: First Workshop on Temperature Aware Computer Systems TACS-01, 2004.

[72] M.-N. Sabry, Dynamic compact thermal models: an overview of current and potential advances, in: International Workshop on Thermal Investigations of ICs (THERMINIC), October, 2002, pp. 1–18.

[73] M.M. Sabry, J.L. Ayala, D. Atienza, Thermal-aware compilation for system-on-chip processing architectures, in: 20th Great lakes symposium on VLSI, May, 2010.

[76] L. Shang, R.P. Dick, Thermal crisis: challenges and potential solutions, IEEE Potentials 25 (September) (2006).

[77] B. Shi, Y. Zhang, A. Srivastava, Dynamic thermal management for single and multicore processors under soft thermal constraints, in: International Symposium on Low-Power Electronics and Design (ISLPED), 2010, pp. 165–170.

[80] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, D. Tarjan, Temperature-aware computer systems: opportunities and challenges, IEEE Micro 23 (6) (2003) 52–61.

[82] K. Skadron, M.R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, D. Tarjan, Temperature-aware microarchitecture: modeling and implementation, in: ACM Transactions on Architecture and Code Optimization (TACO), vol. 1, March, 2004, pp. 94–125.

[83] J. Srinivasan, S.V. Adve, Predictive dynamic thermal management for multimedia applications, in: 17th Annual International Conference on Supercomputing, June, 2003.

[94] S. Zhang, K.S. Chatha, Approximation algorithm for the temperature-aware scheduling problem, in: IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2007, pp. 281–288.

Israel Koren is currently a Professor of Electrical and Computer Engineering at the University of Massachusetts, Amherst and a fellow of the IEEE. He has been a consultant to numerous companies including IBM, Analog Devices, Intel, AMD and National Semiconductors. His research interests include Fault-Tolerant systems, Computer Architecture, VLSI yield and reliability, Secure Cryptographic systems, and Computer Arithmetic. He publishes extensively and has over 200 publications in refereed journals and conferences. He is an Associate Editor of the VLSI Design Journal, and the IEEE Computer Architecture Letters. He served as General Chair, Program Chair and Program Committee member for numerous conferences. He is the author of the textbook "Computer Arithmetic Algorithms," 2nd Edition, A.K. Peters, 2002, a co-author of "Fault Tolerant Systems," Morgan-Kaufman, 2007, and an editor/co-author of "Defect and Fault-Tolerance in VLSI Systems," Plenum, 1989.

C.M. Krishna received his PhD from the University of Michigan. Since then, he has been at the University of Massachusetts, where he is now a professor of electrical and computer engineering. He has coauthored books on real-time systems and fault-tolerant computing. His research areas include real-time systems, fault-tolerance, distributed systems, and sensor networks.