PROMETHEUS: A Proactive Method for Thermal Management of Heterogeneous MPSoCs

Shervin Sharifi, *Member, IEEE,* Dilip Krishnaswamy, *Senior Member, IEEE,* and Tajana Šimunić Rosing, *Member, IEEE*

***Abstract*—In this paper, we propose *PROMETHEUS*, a framework for proactive temperature aware scheduling of embedded workloads on single instruction set architecture heterogeneous multiprocessor systems-on-chip. It systematically combines temperature aware task assignment, task migration, and dynamic voltage and frequency scaling. *PROMETHEUS* is based on our novel low overhead temperature prediction technique, *Tempo*. In contrast to previous work, *Tempo* allows accurate estimation of potential thermal effects of future scheduling decisions without requiring any runtime adaptation. It reduces the maximum prediction error by up to an order of magnitude. Using *Tempo*, *PROMETHEUS* framework provides two temperature aware scheduling techniques that proactively avoid power states leading to future thermal emergencies while matching the performance needs to the workload requirements. The first technique, *TempoMP*, integrates *Tempo* with an online multiparametric optimization method to guide decisions on task assignment, migration, and setting core power states in a temperature aware fashion. Our second scheduling technique, *TemPrompt* uses *Tempo* in a heuristic algorithm that provides comparable efficiency at lower overhead. On average, these two techniques reduce the lateness of the tasks by 2.5× and energy-lateness product (ELP) by 5× compared to the previous work.**

***Index Terms*—Multiprocessor system-on-chip, scheduling, temperature, thermal management.**

I. Introduction

Continuous increase of power density in modern processors results in higher temperatures that lead to system reliability degradation, leakage power increase, performance degradation, and higher cooling and packaging costs. Temperature has become one of the major factors in design, manufacturing, and test of modern multiprocessor systems. Heterogeneous multiprocessor systems-on-chips (MPSoCs) provide tradeoffs regarding performance, power, and temperature by allowing customization of performance and power characteristics of the chip to match the requirements of the workload [1]. Temperature is of particular concern in heterogeneous MPSoCs due to the inherent imbalance in heat generation patterns across the die. These MPSoCs are used in many embedded systems that experience a wider range of environmental conditions than typically seen in data centers and offices without the benefit of more sophisticated packaging due to cost and space considerations. For such systems, on-chip thermal and power management techniques are key.

In this paper, we propose a framework called *PROMETHEUS* to address dynamic thermal management (DTM) of single instruction set architecture (ISA) heterogeneous MPSoCs running embedded workloads. By single ISA heterogeneous MPSoC (assymetric multicore processor), we specifically refer to the SoCs consisting of cores that support the same ISA, but provide different operating points in terms of power and performance. In such systems, cores of different types are able to run the same binary and tasks can migrate between different types of cores. However, different types of cores are optimized for different objectives. Typically one type is optimized for performance while another type may be optimized for power and energy efficiency. Our methodology allows systematic thermal management of these heterogeneous MPSoCs considering individual performance, power, and thermal characteristics of each core. To evaluate the thermal impact of the potential power state changes, we propose a temperature prediction method, called *Tempo*, which has a number of advantages over previous temperature predictors. First, it does not need any kind of runtime adaptations to assure accurate prediction. Moreover, other predictors need to wait until the thermal effect of the power state changes appear in the temperature, so that the predictor can detect the new trend and restart prediction based on that. However, *Tempo* is able to predict temperature of any set of future power state choices before they are applied to the system. This enables accurate upfront evaluation of future thermal impact of potential power state changes caused by scheduling decisions in an MPSoC. Moreover, other methods relying on signal estimation techniques typically treat temperatures of different cores as independent signals. Therefore, in the cases where mutual thermal effects of the cores are significant, inaccuracies in the estimates might be large.

We use *Tempo* as a part of two scheduling techniques to verify thermal safety of alternative scheduling decisions. The first technique, *TempoMP*, integrates *Tempo* in a multiparametric optimization framework. Given the thermal state of the cores and performance requirements of the power states are chosen

Manuscript received November 30, 2011; revised April 13, 2012 and August 31, 2012; accepted October 11, 2012. Date of current version June 14, 2013. This work was supported in part by NSF SHF #0916127, Qualcomm, MuSyC, and SRC #2169.001. This paper was recommended by Associate Editor Yuan Xie.

S. Sharifi and D. Krishnaswamy are with Qualcomm Technologies, Inc., San Diego, CA 92121 USA (e-mail: ssharifi@qualcomm.com; dilip@ieee.org).

T. Š. Rosing is with the Computer Science Department, University of California, San Diego, CA 92093 USA (e-mail: tajana@cse.ucsd.edu).

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

Digital Object Identifier 10.1109/TCAD.2013.2247656

that are power efficient, thermally safe, and able to provide required performance for the workload. Our second technique, *TemPrompt* also uses *Tempo* to estimate the future thermal effects of various scheduling decisions, but instead of being optimal, it uses a heuristic algorithm to choose power state assignments. These techniques are preemptive multitasking scheduling techniques that can preempt the tasks at the end of each time quantum (at each scheduling tick) or migrate them to other cores. Compared to previous work, these two techniques provide on average a 2.5× reduction in lateness of the tasks and 5× reduction in energy-lateness product. The next section discusses the related work followed by the sections describing the details of *Tempo* and *PROMETHEUS*.

II. Related Work

Availability of multiple instances of similar processing resources on MPSoCs creates further opportunities for thermal management compared to single core processors by allowing distribution of activities and heat as necessary. However, the solution space becomes huge due to the multitude of possibilities regarding assignment of power states and tasks to the cores and deciding about when to start the available tasks. The problem is even more complicated by the fact that temperature of any particular point on the die is a strong function of recent temperature and workload history. In general, task scheduling under thermal constraints is an NP-hard problem.

A large number of DTM techniques proposed in the past have been reactive in the sense that they take action when temperature of a unit rises above a given threshold. This might cause significant performance loss and reliability issues [2]. Therefore, proactive DTM techniques have been proposed that try to predict and prevent thermal emergencies before they happen. Several temperature predictors have been proposed to be used for proactive DTM techniques. In [3] a proactive temperature balancing technique based on an autoregressive moving average (ARMA) modeling has been proposed for general-purpose systems. In order to avoid inaccuracy, the changes in the workload and temperature dynamics are detected using a sequential probability ratio test (SPRT), so that the ARMA model can be updated in timely fashion. In systems with highly dynamic workloads, continuously performing SPRT-based detection and updating the ARMA model incurs overhead.

In [4], a temperature prediction technique for chip multiprocessors is proposed that assumes future temperature of a cores as the linear extrapolation of its previous temperature readings. Although this predictor does not need any kind of runtime adaptation, because it implicitly assumes that the gradient of temperature stays the same for the whole prediction interval, it may result in inaccurate estimates.

In [5] a proactive DTM technique is introduced for chip multiprocessors based on the band-limited property of temperature frequency spectrum. Although this method does not need any adaptation either, it is not able to accurately predict temperature changes due to power changes before they actually happen. This technique predicts the temperature by observing the previous samples and projecting them into future assuming the trend does not change. Therefore, when the power state of a core changes, this technique needs to observe at least a few samples before being able to estimate the future temperature. Thus it cannot be used to evaluate potential thermal impact of alternative future scheduling decisions.

Previous temperature prediction techniques either need costly runtime adaptation (e.g., [3]) or are not able to accurately predict the thermal impact of transition to new power states before the power change happens (e.g., [5]). The general signal analysis and prediction approaches used in these techniques are more suitable for the cases where the underlying physical model of the system is not well known. Being oblivious to this knowledge of the system, the advantages of these techniques are limited when the underlying model of the system is known because they are not getting the full benefit of this valuable information at hand. In contrast to previous techniques, our prediction method takes advantage of the knowledge of the dynamics of the system, which is provided by the thermal model. Unlike previous techniques, *Tempo* can accurately predict what the future temperature of the cores can be for any future power states of the cores without the need to apply them and wait to see their effect on temperature. Therefore, it can be used to evaluate the thermal effects of alternative decisions for thermal management to choose the best out of potential options. Moreover, our proposed model does not need any runtime adaptation and also is fully linear.

Tempo can be used within model predictive control techniques as well. DTM techniques using model predictive control such as [6] typically assume the temperatures at all of the nodes of the thermal network are observable by thermal sensors, which is not true for majority of designs because thermal sensors are not placed in thermal interface material and heat sink. Instead, *Tempo* can be used in such cases in order to avoid such assumptions, especially because it is completely linear so there is no need for non-linear optimization techniques. It can also be easily integrated with multiparametric optimization techniques, as we have used in *TempoMP*, which is explained later. *Tempo* can also be used with phase detection and prediction techniques such as [7], which are able to monitor and predict power consumption of the cores. Although availability of power estimates increases prediction accuracy of *Tempo*, there are no accurate values of future core power consumptions, *Tempo* is able to estimate future temperature based on only previous temperature information.

Some recent work leverages control theory and optimization to manage thermal issues in homogeneous MPSoCs. In [8], convex optimization is used to control the frequency of the cores on a homogeneous MPSoC to guaranty that thermal constraints are met. In [9], a linear quadratic regulator is used to solve the frequency assignment problem for thermal balancing. To achieve a smooth control and to minimize performance loss and thermal fluctuations in an MPSoC, [6] proposes a technique based on model predictive control.

Unlike general-purpose processing, the type and characteristics of the workloads such as execution time are often known in advance in embedded domain. Although this extra information is helpful in devising better thermal management solutions, the complexity of the problem makes it practically infeasible to find the optimal solution. To manage this complexity and make

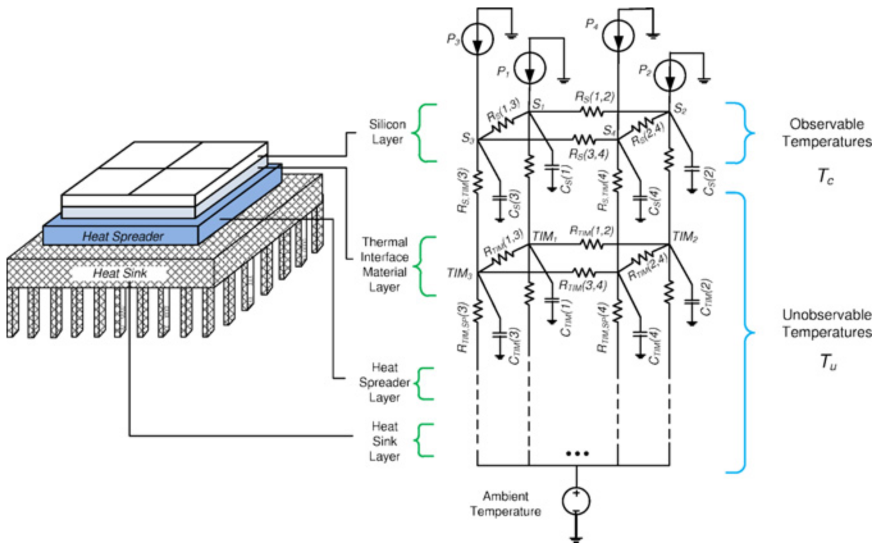


Fig. 1. Example of thermal network for an MPSoC with four cores.

the problem tractable, various techniques in this domain have used different simplifying assumptions to develop heuristics. In contrast to general-purpose domain, in embedded domain the goal is not merely maximizing the throughput and each task might have individual performance requirements such as deadlines. In [10], an assignment and scheduling technique for hard real-time applications on MPSoCs is proposed that uses a mixed-integer linear programming formulation to minimize peak temperature under hard real-time constraints and task dependencies. This technique is limited to tasks with large execution times and works based on a steady-state thermal model. It performs global optimization to minimize the temperature of a set of known tasks. The complexity of this approach increases exponentially with the number of tasks and number of cores, and as authors have mentioned, this formulation is not practical for large problem instances. A heuristic approach is also presented for large problems, which does not consider a thermal model and works based on the mobility of the tasks.

Relatively little work has focused on heterogeneous embedded MPSoCs. The work in [11] proposes an algorithm for energy and temperature aware scheduling of embedded workloads on heterogeneous MPSoCs where the scheduler operates in two modes based on the utilization of MPSoC. The workload is observed and its performance requirements are estimated. Then, based on the utilization of the processor it is decided if the scheduler should work in energy-saving or thermal management mode.

In this paper, we propose *PROMETHEUS* framework for proactive management of temperature in heterogeneous MPSoCs by systematically selecting power state of the cores and task assignments. *PROMETHEUS* provides two scheduling techniques, *TempoMP* and *TemPrompt*, which both are based on our temperature prediction technique, *Tempo*. *TempoMP* incorporates *Tempo* temperature prediction with multiparametric optimization to choose the optimal alternative among possible power states of the cores and task assignments that does not violate thermal requirements. The other technique, *TemPrompt* uses *Tempo* within a heuristic algorithm that provides comparable efficiency to *TempoMP*, but at a lower overhead.

PROMETHEUS framework is general and is able to consider various performance, power, and thermal characteristics for cores, which makes it applicable to heterogeneous MPSoCs as well as homogeneous ones. It meets thermal requirements by evaluating future impacts of various scheduling decisions and avoiding decisions leading to thermal emergencies.

The next section explains the details of *Tempo* temperature prediction technique. Section IV discusses our scheduling framework, *PROMETHEUS*, and details of our proposed scheduling techniques. Section V provides the experimental results and discussions, and Section VI concludes this paper.

III. Temperature Prediction

In this section, we first explain the basic ideas behind our *Tempo* temperature predictor and then we describe how *Tempo* can take into account the leakage and its temperature dependence. The objective of our prediction method is to accurately predict future temperature of the cores based on the available temperature and power information. More specifically, we estimate the temperature of the cores at the end of the scheduling tick $(k + 1)$ based on the temperature of the cores at the beginning of scheduling tick $k + 1$ and the one before that (k). *Tempo* takes into account the power state changes between scheduling ticks $k + 1$ and k as well.

Our work is based on compact thermal model of the chip [2] that leverages the well-known duality of thermal and electrical phenomena. The heat flow among the functional units is modeled using a corresponding network of thermal capacitances and resistances as shown in Fig. 1. The dynamics of the temperature and the relation between the temperature, power consumption of the cores, and thermal characteristics of the system are described as

$$C_t \frac{\mathrm{d}}{\mathrm{d}t} T(t) = -G_t T(t) + P(t) \tag{1}$$

where the vectors and matrices are defined as:

1) T temperature at all the nodes of the thermal network;
2) P power consumptions of the nodes of thermal network;

3) G_t thermal conductance matrix;
4) C_t thermal capacitance matrix.

Assuming the number of the cores to be n and the number of nodes in the thermal RC network to be m, P and T are vectors of length m both and G_t and C_t are matrices of size $m \times m$ both.

Given the temperature at all the nodes of the thermal network, estimating the future temperature based on the power is not difficult. However, usually this is not the case. At runtime, temperature is usually obtained via thermal sensors within the silicon layer. If each core does not have its own sensor, the technique in [12] can be used to estimate the core temperatures using the available sensors. However, thermal sensors cannot be placed within internal layers (thermal interface material, heat spreader, etc.), so the available temperature information is limited to the temperature of the cores. This lack of thermal information of internal nodes makes it challenging to predict or evaluate temperature at runtime. The temperature of the internal nodes can be obtained by simulating the thermal model at runtime, which is not computationally practical. Moreover, without feedback from the thermal sensors, the results may deviate dramatically from the actual values.

In our formulation, to reflect this lack of thermal information of the internal nodes, we break the vector of temperature values (T) into two subvectors. Subvector T_o represents the temperatures observable by thermal sensors (core temperatures), while T_u represents the internal nodes of the thermal network whose temperatures are unobservable (as shown in Fig. 1). The size of these vectors are n and $m-n$ respectively.

The analytical solution to the non-homogeneous system of differential equations in (1) can be calculated as

$$T(t) = e^{-\Gamma(t-t_0)}T(t_0) + \int_{t_0}^{t} e^{-\Gamma(t-\tau)}C_t^{-1}P(\tau)\,\mathrm{d}\tau \tag{2}$$

where $T(t_0)$ is the starting temperature at time t_0 and

$$\Gamma = C_t^{-1}G_t. \tag{3}$$

Discretizing (2) for a scheduling tick of t_s, the temperature at consequent scheduling ticks can be described as

$$\begin{bmatrix} T_o[k+1] \\ T_u[k+1] \end{bmatrix} = \Psi \begin{bmatrix} T_o[k] \\ T_u[k] \end{bmatrix} + \Phi \begin{bmatrix} P[k+1] \\ P_u[k+1] \end{bmatrix} \tag{4}$$

where

$$\Psi = e^{-\Gamma t_s},\ \Phi = \Gamma^{-1}(I - e^{-\Gamma t_s})C_t^{-1}. \tag{5}$$

The first term in (4) is the contribution of initial conditions, and the second term is the contribution of power consumption during this scheduling tick. We divide the matrices Ψ and Φ into submatrices as shown here

$$\Psi = \left[\begin{array}{c|c} \Psi_{oo} & \Psi_{uo} \\ \hline \Psi_{ou} & \Psi_{uu} \end{array}\right],\ \Phi = \left[\begin{array}{c|c} \Phi_{oo} & \Phi_{uo} \\ \hline \Phi_{ou} & \Phi_{uu} \end{array}\right] \tag{6}$$

where sizes of the matrices Ψ_{oo}, Ψ_{uo}, Ψ_{ou}, and Ψ_{uu} are $n \times n$, $n \times m-n$, $m-n \times n$, and $m-n \times m-n$, respectively. Each matrix Ψ_{xy} shows the effect of initial temperature of set x of nodes on the current temperature of the set y. For example, Ψ_{uo} models the effect of initial temperature of *unobservable(u)* nodes on the current temperature of *observable(o)* nodes. Similarly, each matrix Φ_{xy} shows the effect of current power of the set x of nodes on the current temperature of the set y. The internal nodes do not consume any power ($P_u[k+1] = 0$), so

$$T_o[k+1] = \underbrace{\Psi_{oo}T_o[k]}_{F_o} + \underbrace{\Psi_{uo}T_u[k]}_{F_u} + \underbrace{\Phi_{oo}P[k+1]}_{F_p} \tag{7}$$

where the first and second terms (F_o and F_u) are respectively the contributions of initial temperature of the observable and unobservable nodes on the temperature at the next scheduling tick. The third term (F_p) is the contribution of the power consumption of the cores during the incoming scheduling tick. At each scheduling tick, the term F_o can be calculated based on the current temperature of the cores which are observable by thermal sensors. The term F_p also can be calculated given the current power consumption of the cores. But due to lack of knowledge about the temperature of unobservable nodes ($T_u[k]$), term F_u is unknown. This term represents the contribution of initial temperature of the unobservable nodes of the thermal network (internal nodes) on the temperature at the next scheduling tick. Because F_u is not known at runtime, (7) is not enough to calculate future temperature of the cores.

We show that fast changes in the temperature are produced by components F_o and F_p while component F_u does not change quickly. Fig. 2 shows an example of breakdown of the temperature of a SPARC-like core in floorplan of Fig. 7

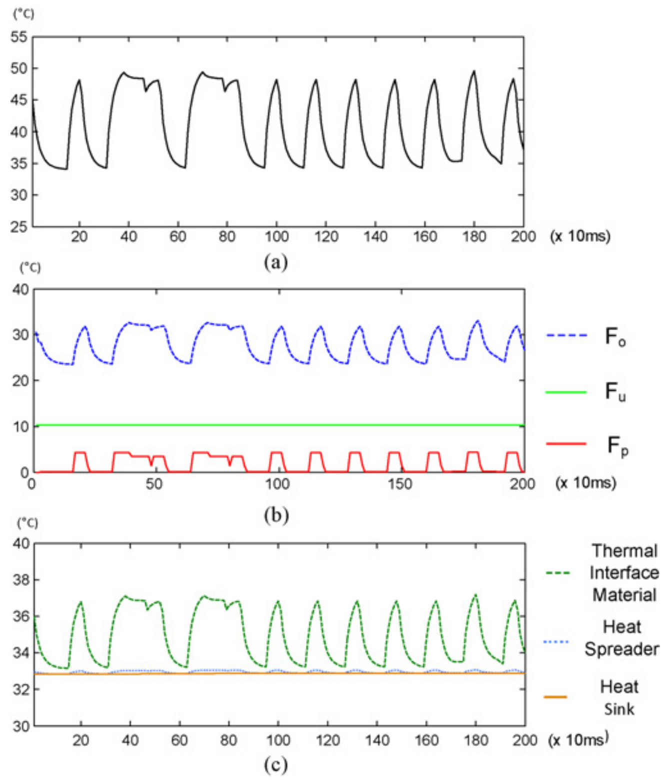


Fig. 2. (a) Temperature of the core. (b) Breakdown of temperature into components of (7). (c) Temperature of corresponding nodes in thermal interface material, heat spreader, and heat sink, all relative to ambient.

into three components of (7). It should be noted that the core temperature shown in part (a) of the figure is the sum of the components in Fig. 2(b). All values are relative to the ambient temperature (40 °C in this case). As can be seen in this figure, although the core temperature changes significantly, the term F_u changes very slowly. The quick change of temperature between two scheduling ticks is mainly due to changes on the other two terms (F_o and F_p) as shown in the figure. Fig. 2(c) shows the temperature of the corresponding internal nodes. It should be noted that as this figures shows, although some of the unobservable internal nodes might change quickly (e.g., thermal interface material), the changes of term F_u are very slow. Later in this section we will explain the reason for this phenomenon in detail.

Due to the limited rate of change on F_u, we assume that this term remains constant between two consecutive scheduling ticks

$$F_u[k+1] \approx F_u[k]. \tag{8}$$

Temperature evaluation equation for the previous scheduling tick is then

$$T_o[k] = \Psi_{oo}T_o[k-1] + \Psi_{uo}T_u[k-1] + \Phi_{oo}P[k]. \tag{9}$$

Based on (8), $\Psi_{II}T_u[k] \simeq \Psi_{II}T_u[k-1]$. $\mathbb{T}$, the temperature prediction by *Tempo* is then calculated as

$$\mathbb{T}_o[k+1] = \underline{(\Psi_{oo}+I)T_o[k] - \Psi_{oo}T_o[k-1]} + \Phi_{oo}(P[k+1] - P[k]). \tag{10}$$

We use (10) to predict the temperature at the beginning of scheduling tick $k+1$ based on the temperature of the cores at the beginning of scheduling tick k and their power state during this scheduling tick. We also define *Tempo's* thermal state of a core, $\mathcal{T}[k+1]$ in (11) as the predicted temperature at the beginning of scheduling tick $k+1$ if the power state of the cores do not change in scheduling tick k.

$$\underline{\mathcal{T}[k+1]} = (\Psi_{oo}+I)T_o[k] - \Psi_{oo}T_o[k-1]. \tag{11}$$

As (11) shows, $\mathcal{T}[k+1]$ can be calculated based on the previous and current core temperatures. Later we use *Tempo*'s thermal state within our scheduling techniques. Based on (11) we rewrite (10) as

$$\mathbb{T}_o[k+1] = \underline{\mathcal{T}[k+1]} + \Phi_{oo}(P[k+1] - P[k]). \tag{12}$$

The second term on the right-hand side reflects the effect of power changes on the temperature of the cores.

Now we will describe how we incorporate the leakage power and its temperature dependence into *Tempo* prediction. We use the linear approximation of the leakage as suggested in [13] with an approximate estimation error of up to 5%. Using this model, the leakage power of a core can be estimated as sum of a constant term and a term linearly dependent on the cores temperature

$$P_{leak}(t) = LT(t) + Q \tag{13}$$

where L is a diagonal matrix containing the coefficients for the linear terms and Q is a vector of constant terms for different cores. It should be noted that elements of L and Q, which correspond to the nodes in any layer other than silicon, are zero because these nodes do not consume any power, including leakage power. Therefore, (1) is transformed to

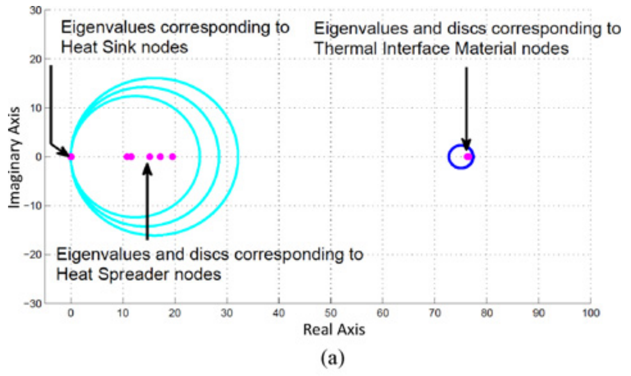
$$C_t\frac{\mathrm{d}}{\mathrm{d}t}T(t) = -(G_t - L)T(t) + (P_{\text{dynamic}}(t) + Q). \tag{14}$$

The next steps would be similar to deriving equations (2) to (12) based on (1). Predicted temperature considering leakage is

$$\mathbb{T}_o[k+1] = \big((\Psi'_{oo}+I)T_o[k] - \Psi'_{oo}T_o[k-1]\big) + \Phi'_{oo}(P_{\text{dyn}}[k+1] - P_{\text{dyn}}[k]) \tag{15}$$

where $\Psi' = e^{-C_t^{-1}(G_t-L)t_s}$, $\Phi' = ((G_t - L)^{-1}C_t)(I - e^{-C_t^{-1}(G_t-L)t_s})C_t^{-1}$.

Before describing how we leverage *Tempo* in our scheduling framework, which is explained in Section IV, in the following subsection we show why *Tempo* is an accurate predictor provide a theoretical upper bound on the prediction accuracy of *Tempo*.



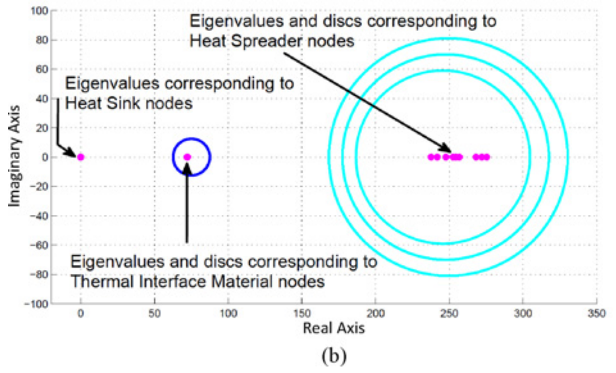


Fig. 3. Gershgorin discs of matrix Γt_s in complex plane for (a) high-end package and (b) embedded-type package.

A. *Theoretical Analysis of Tempo*

Tempo is based on the premise that the changes on component F_u of temperature are very slow, or $F_u[k+1] \approx F_u[k]$. This phenomenon can be explained by the structure of the thermal RC circuit and thermal characteristics of the chips. To simplify the explanation, we assume that the length and width of heat spreader and heat sink exactly match those of the silicon (and thermal interface material).

As shown in Fig. 1, each node in the thermal network is connected to its neighbor nodes in the same layer through lateral thermal conductances and to the corresponding nodes in the top and bottom layers through vertical thermal resistances. If the nodes at the bottom and top of node i are, respectively, represented by $bottom(i)$ and $top(i)$ and the set of lateral neighbors of the node i are represented by $LN(i)$, then thermal conductance matrix G_t could be described as

$$G_t(i,j) = \begin{cases} i = j & \sum_{l \in LN(k)} g(i,l) + g(i, top(i)) + g(i, bottom(i)) \\ i \neq j & -g(i,j) \end{cases} \tag{16}$$

where $g(i,j)$ is the conductance and capacitance between nodes i and j in the thermal circuit. Capacitance matrix C_t is also defined as

$$C_t(i,j) = \begin{cases} i \neq j & 0 \\ i = j & c(i) \end{cases} \tag{17}$$

where $c(i)$ is the thermal capacitance of node i. As can be seen, matrix G_t is symmetric and diagonally dominant, and matrix C_t is diagonal. Based on (16) and (17)

$$\Gamma(i,i) = c(i)^{-1} \left(\sum_{l \in LN(k)} g(i,l) + g(i, top(i)) + g(i, bottom(i)) \right). \tag{18}$$

Considering matrix Γt_s, its eigendecomposition is

$$\Gamma t_s = V \Lambda V^{-1} \tag{19}$$

where V is a square matrix whose jth column is the jth eigenvector (v_j) of Γt_s and the Λ is a diagonal matrix whose diagonal elements are the corresponding eigenvalues of matrix Γt_s; in other words: $\Lambda(j,j) = \lambda_j$. From (5) and (19), we have

$$\Psi = e^{-\Gamma t_s} = V e^{-\Lambda} V^{-1} = V X V^{-1}. \tag{20}$$

Since Λ is diagonal, X is also a diagonal matrix with diagonal elements

$$X(j,j) = e^{-\lambda_j}. \tag{21}$$

According to Gershgorin's circle theorem [14], the eigenvalues of matrix Γt_s lie within a set of discs called Gershgorin discs. Gerschgorin disc i is centered at $\Gamma(i,i)t_s$ with radius $\rho(i)$. $\rho(i)$ is defined as $\rho(i) = min(\rho_R(i), \rho_C(i))$ where

$$\rho_R(i) = \textstyle\sum_{j \neq i} |\Gamma(i,j)t_s| \quad \rho_C(i) = \textstyle\sum_{j \neq i} |\Gamma(j,i)t_s|. \tag{22}$$

For diagonally dominant matrices, usually the eigenvalues tend to be closer to the centers of the discs. For example, in the extreme case of a diagonal matrix, the eigenvalues fall right onto the center of Gershgorin discs. Using (18) and (22), Gershgorin discs and the location of the eigenvalues can be estimated directly based on the chip and package parameters without the need to calculate the exact eigenvalues.

For an MPSoC composed of a 3×3 grid of cores of size 1×1 mm each, Fig. 3 shows the eigenvalues and Gershgorin discs of matrix Γt_s corresponding to the nodes in thermal interface material, heat spreader and heat sink for two different types of packages. The first one is the default package in Hotspot [15] which is a high-end package with thermal interface material thickness of 0.02 mm, heat spreader thickness of 1 mm, heat sink thickness of 6.9 mm and convection resistance of 0.1 K/W. The other one resembles a lower end and less expensive package that can be found in embedded type devices. It has the same thermal interface material thickness, but with heat spreader thickness of 0.1 mm, heat sink thickness of 1 mm, and convection resistance of 5.0 K/W.

In both cases, the eigenvalues of matrix Γt_s corresponding to heat sink (shown in Fig. 3) are very close to zero and their corresponding Gershgorin discs are too small to be seen in the figure. In both types of packages, the eigenvalues corresponding to the thermal interface material and heat spreader are much larger compared to those of heat sink, although the heat spreader eigenvalues are very different in these two cases. As a result, in matrix Ψ, according to (21) the eigenvalues corresponding to heat spreader and thermal interface material are negligible compared to the ones of the heat sink. Therefore, the effect of the temperature of these nodes on component F_u is negligible relative to the effect of the heat sink. Consequently, although the temperature of thermal interface material and heat spreader can change quickly and significantly, the effect of these changes on the component F_u is negligible. In contrast, the temperature of heat sink that is the dominant component of F_u changes very slowly. Its changes are negligible in the milliseconds range. Therefore, F_u's changes between two consecutive scheduling ticks can be neglected.

The next section describes how we leverage *Tempo* in our proposed scheduling framework.

IV. PROMETHEUS SCHEDULING FRAMEWORK

In this section, we describe *PROMETHEUS*, our proposed framework for proactive temperature aware scheduling. Our approach imposes no restrictions on the distributions of the size and the number of concurrent tasks in the workload, which makes it applicable to various classes of workloads. It also systematically considers individual performance, power, and thermal characteristics of the cores so it can be applied to heterogeneous MPSoCs as well as homogeneous ones. Using *Tempo*, the thermal impact of alternative scheduling decisions is evaluated in advance and scheduling decisions, which might result in thermal emergencies, are avoided. Therefore, *PROMETHEUS* guarantees that the maximum temperature threshold will not be exceeded. This approach can be applied to any preemptive, multitasking scheduling system that is able to preempt the tasks or migrate them to other cores at scheduling intervals. Here, we present an overview of *PROMETHEUS's* scheduling system followed by the details of two individual proactive scheduling techniques.

Fig. 4 shows an overall view of our framework and its operation. Embedded workloads running on the system are pre-characterized in terms of execution time and power. Using pre-characterization information, the *performance requirement estimation module* can estimate the execution time of the tasks at different power states. Then, for each core type it can determine the set of power states on which each tasks will be able to meet its performance requirement. For cases where offline characterization of the tasks is not possible,

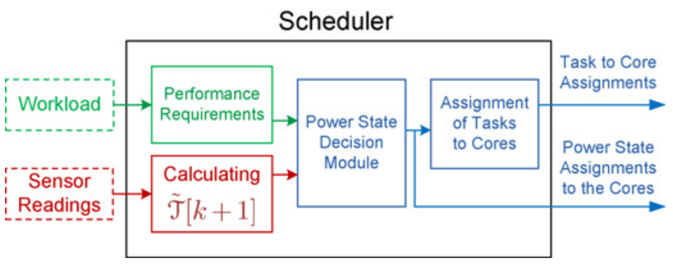

Scheduler
Workload
Sensor Readings
Performance Requirements
Calculating $\tilde{\mathcal{T}}[k+1]$
Power State Decision Module
Assignment of Tasks to Cores
Task to Core Assignments
Power State Assignments to the Cores

Fig. 4. Scheduling system in *PROMETHEUS*.

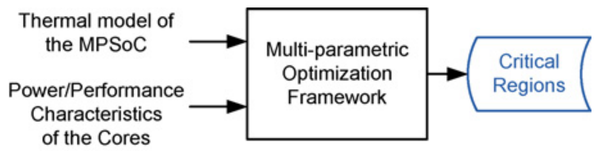

Thermal model of the MPSoC
Power/Performance Characteristics of the Cores
Multi-parametric Optimization Framework
Critical Regions

Fig. 5. Offline stage of *TempoMP*.

online characterization techniques can be used to evaluate the performance requirements on the fly [16].

At each scheduling tick, the *temperature prediction module* calculates *Tempo's* thermal state, $\mathcal{T}[k+1]$, which is the estimated temperature at next scheduling tick if the current power states of the cores do not change. Given the outputs of the *temperature prediction module* and *performance requirement estimation module*, *power state decision module* determines a set of thermally safe power states that are able to provide performance as close as possible to the workload's requirements. The output of this module is used to set the core power states and also in assigning tasks to cores. Once the power states are determined by the *power state decision module*, *task assignment decision module* decides how the tasks should be assigned to the cores based on their performance requirements.

The two scheduling techniques provided in this framework, *TempoMP* and *TemPrompt*, differ only in their *power state decision module*s. The first technique, *TempoMP*, determines the safe power states based on an optimization stage that is performed offline and its results are stored for runtime use. Although the optimization technique can provide locally optimal power state decisions, storing and fetching the optimization results incurs overhead. To avoid this overhead, we propose another scheduling technique called *TemPrompt* that uses a heuristic for its *power state decision module*. The next subsection describes the details of *TempoMP* and *TemPrompt* scheduling techniques.

A. Power State Assignment in TempoMP

At each scheduling tick, temperature information received from the sensors and also scheduling state provided by the scheduler are used in order to determine the core power states and task assignments for the next scheduling interval. In *TempoMP*, power state module determines the best power state for the cores by referring to the results of offline optimization stage. Optimization is formulated based on the power and performance characteristics of the cores and the thermal model of the MPSoC as shown in Fig. 5. The selection of locally optimal power states and task assignment are done based on a multiparametric optimization framework that incorporates our temperature prediction methods.

Multiparametric programming is a class of optimization problems in which given an objective function, a set of constraints, and a set of parameters, the optimal solution is obtained as an explicit function of those parameters [17]. It is a powerful approach for analyzing the effect of variations and uncertainty in optimization problems where objective function is to be minimized or maximized subject to a set of constraints, and a set of parameters that may vary within given bounds. For such problems, multiparametric programming obtains the objective function and optimization parameters as functions of the varying parameters. It also provides the regions in the solution space where these functions are valid. These regions are called *critical regions*. Multiparametric optimization approach splits the optimization process into offline and online stages. Using multiparametric programming, the optimization problem is solved offline and the set of *critical regions* and parametric solutions are provided as output. No optimization needs to be done at runtime. Only a limited number of operations need to be performed at runtime in order to find the *critical regions* corresponding to the current solution.

We use this technique as a basis for our online optimization. Our goal is to choose what power states the cores should be set to in order to meet the performance requirements of the workload and thermal constraints. Therefore, decision variables (optimization outputs) are power states of the cores while performance requirements and the thermal state of the MPSoC are the varying parameters (parametric optimization inputs) in this optimization problem. Without this online optimization, various combinations of power states would have to be tried to find the thermally safe power settings which meet the performance requirements of the workload. Performance requirement of a workload is measured by the minimum number of cores of type Ω that should be set to a given power state v to meet performance demand of the workload and is denoted as $\sigma_{\Omega,v}$. Performance requirement of a task is measured by the time to its deadline in a deadline based system. In other types of systems, per-task performance requirements can be described using other measures such as throughput requirements.

Fig. 6 shows a simple illustrative example of using multiparametric programming in power state assignment for a single core case. This core has one sleep state and three discrete power states: p_{high}, p_{medium}, and p_{low} (corresponding to frequencies f_{high}, f_{medium}, and f_{low}, respectively). The goal is to set the core to the optimal power state whose frequency is equal to or greater than the minimum frequency required to meet performance requirements of the workload (f_{min}) but does not cause the temperature to exceed the threshold. f_{min} is one of the varying parameters in the optimization and its value is estimated at runtime given the workload. Another parameter obtained at runtime in this decision is *Tempo's* thermal state of the core, $\mathcal{T}[k+1]$, which is the estimated temperature of the core at the next scheduling tick assuming the power state does not change. $\mathcal{T}[k+1]$ reflects the current trend of the temperature of the cores.

This set of parameters defines the solution space as shown in Fig. 6. The y-axis represents f_{min}, while the x-axis represents $\mathcal{T}[k+1]$. The space of the solutions consists of all possible combinations for $\mathcal{T}[k+1]$ and f_{min}, which is divided into four regions. Based on the given input parameters, the corresponding region is identified and the corresponding solution is chosen. Each region corresponds to one frequency setting that minimizes the power while providing at least the minimum required frequency. In the example of Fig. 6, each region is labeled by the frequency corresponding to that region.

At very low $\mathcal{T}[k+1]$, even the highest power state is thermally safe. Therefore, even if the frequency requirement (f_{min}) is high, it can be satisfied. This corresponds to the green area of Fig. 6. When $\mathcal{T}[k+1]$ is very high, none of the active states are thermally safe, therefore the core should be put to sleep (e.g., when $\mathcal{T}[k+1] > 86\,°\text{C}$). This corresponds to the white region on the right side of the figure. At $\mathcal{T}[k+1]$ values between these two extremes, the core can be set to one of its active power states. As an example, parameters $\mathcal{T}[k+1] = 80\,°\text{C}$ and minimum frequency requirement of 1.1 GHz fall into the yellow region that corresponds to the medium power state of the core (p_{med} corresponding to f_{med}). Using multiparametric programming, we generate such functions that allow us to find optimum power states for each combination of parameters. This example was just a simple illustration of how multiparametric optimization can be used to assign power state to a single core based on its thermal state and the performance demand of the workload. Next we describe how we extend this idea to the more general case of systems with multiple cores and different representation of performance requirements.

We formulate an optimization problem using *Tempo* prediction model with the goal of finding the locally optimal power states of the cores. Thermal states of the cores and performance requirements of the workload are given to the optimization framework as inputs, while the output of the optimization process is a set of power states for the cores that are thermally safe and are able to provide the performance requirements of the workload at the minimum power cost. The optimization formulation uses *Tempo* to evaluate thermal safety of the potential power states.

The decision variables in this optimization are power states in the next scheduling tick that are represented by the vector $\alpha[k+1]$ (in bold in (23)). $\lambda_{\Omega,v}$ represents the number of cores that would be set to power state v after the optimization while $\sigma_{\Omega,v}$ is a measure of performance requirement which represents the minimum number of cores which should be set to power state v to satisfy the performance demand of the workload. The optimization problem is formulated as

$$\begin{aligned} &\underset{\alpha}{\text{minimize}} \quad P_{\text{total}}(\boldsymbol{\alpha}[\boldsymbol{k+1}], \mathcal{T}_o[k+1]) \\ &\text{subject to } \mathbb{T}_o(\boldsymbol{\alpha}[\boldsymbol{k+1}], \alpha[k], \mathcal{T}_o[k+1]) \prec T_{Th} \\ &\qquad\qquad \forall \Omega, v : \lambda_{\Omega,v} \geq \sigma_{\Omega,v}. \end{aligned} \tag{23}$$

where we use $\prec$ as element-wise less than operator and $\mathcal{T}[k+1]$ is the thermal state defined by *Tempo*. The objective of this optimization is to minimize the total power (P_{total}) of the cores under the thermal limits. P_{total} is the sum of dynamic and leakage power of the cores. Leakage component of P_{total} is calculated using (13) and (14).

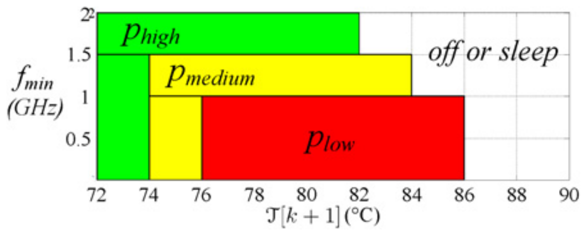


Fig. 6. Very simple example describing use of multiparametric programming in power state assignment.

The optimization parameters are performance requirements and the temperature of the cores. Details of the optimization objective and constraint formulation are provided below.

The outputs of the optimization (the decision variables) are the power states of the cores in the next scheduling tick that are represented by the vector $\alpha[k+1]$. If the optimization sets the core n to power state v, then $\alpha_{n,v}$ is 1, and 0 otherwise

$$\alpha_{n,v} = \begin{cases} 1, & \text{if core } n \text{ is set to power state } v \\ 0, & \text{otherwise.} \end{cases} \tag{24}$$

Optimization chooses $\alpha[k+1]$ values such that the total power is minimized under the thermal constraints while meeting performance requirements. We denote the number of cores of type Ω that are set to power state v as $\lambda_{\Omega,v}$, where

$$\lambda_{\Omega,v} = \sum_{\omega \in \Omega} \alpha_{\omega,v}. \tag{25}$$

The minimum number of cores of type Ω that have to run at a given power state v to meet performance requirements of the workload is denoted as $\sigma_{\Omega,v}$. The *performance requirement estimation module* determines $\sigma_{\Omega,v}$ values at runtime given the performance requirements of the tasks provided by information such as deadlines, required throughput or target IPS (instructions per second) [16] for example. As an example, suppose a soft deadline based system with three ready to run tasks. Based on the current deadlines of the tasks and their pre-characterization information, the *performance requirement estimation module* determines that in order to meet the deadlines of these tasks, two of them need to run on cores of type 0 ($\Omega = 0$) at the highest power state ($v = 0$) and the remaining one needs to be run on a core of type 1 ($\Omega = 1$) at its lowest power state ($v = 2$), then $\sigma_{0,0} = 2$, $\sigma_{1,2} = 1$ and for other values of Ω and v, $\sigma_{\Omega,v} = 0$. For each core, if we assume core of type k has v active power states and one sleep state, the power consumption of core n that is of core type k can be written as

$$P[n] = \alpha_{n,1} \cdot P_{k,1} + \cdots + \alpha_{n,v} \cdot P_{k,v} + \alpha_{n,sleep} \cdot P_{k,sleep} \tag{26}$$

where $P_{k,v}$ is the power consumed at power state v at a core of type k.

The first constraint enforces the predicted temperature at the next scheduling tick to be lower than the maximum threshold, while the second set of constraints require the power states chosen by optimization to provide at least the performance required by the workload. Instead of solving the optimization problem at each scheduling tick, we use an approach based on multiparametric programming [17]. Optimization parameters σ, $\alpha[k]$ and $\mathcal{T}[k+1]$ partition the parameter space into separate regions called *critical regions*. Each possible combination of σ, $\alpha[k]$ and $\mathcal{T}[k+1]$ corresponds to one and only one of these *critical regions* that represents the optimum power states of the cores for that specific combination. The region basically specifies the validity range of that set of power states such that temperatures of all the cores are below the threshold temperature and the total power is minimized. The actual values for the optimization parameters are found at runtime. Given the parameter values, the corresponding region is found that represents the appropriate set of power states for the cores. The set of optimal solutions ($\alpha[k+1]$) is obtained as an explicit function of the parameters (σ, $\alpha[k]$ and $\mathcal{T}[k+1]$). To get the optimum results at runtime, the *power state decision module* does not need to do any optimization online. It only needs a limited number of operations to be performed at runtime to find the regions representing the $\alpha[k+1]$ values corresponding to the current value of σ, $\alpha[k]$ and $\mathcal{T}[k+1]$.

Temperature of the cores at the next scheduling tick depends on decision variable $\alpha[k+1]$ and optimization variables $\alpha[k]$, $\mathcal{T}[k+1]$. Larger number of optimization parameters results in much larger solution space. In order to reduce the number of optimization variables, (10) is transformed to

$$\mathbb{T}_o[k+1] = \underbrace{\Phi_{oo}P[k+1]}_{F_p} + \underbrace{(\mathcal{T}[k+1] - \Phi_{oo}P[k])}_{F_r} \qquad (27)$$

It can be seen that at each scheduling tick, the only unknown component is F_p and the remaining component, F_r, is known. Therefore, at each scheduling tick, F_r can be calculated based on the observable temperature and current power states, and is used as the optimization parameter instead of $\alpha[k]$ and $\mathcal{T}[k+1]$. This significantly reduces the size of the parameter space. F_r along with performance requirements $\sigma_{\Omega,v}$ are used by the *power state decision module* to find the corresponding critical region that represents the set of optimal power states of the cores for the given parameters.

We store the results of optimization in a two-level look-up table. At the first level, for each combination of performance requirements ($\sigma_{\Omega,v}$), one set of regions is stored. At the second level, within that set of regions, we find the proper region based on F_r. Performance requirements ($\sigma_{\Omega,v}$ values) are small integer values (the upper bound for $\sigma_{\Omega,v}$ is the number of cores of type Ω). Therefore, given the values of variable $\sigma_{\Omega,v}$, the table where the corresponding set of regions is stored can be found with a single memory access. Then the F_r values are used to find the proper region within this set. If n_r is the average number of regions in each set of regions and n_c is the number of the cores, then finding the right region within a set needs $n_r \times n_c$ multiply/add operations on average [17]. Therefore, the complexity of this algorithm is $O(n_r \times n_c)$ ($n_r < 5$ in our experiments).

After the power states are set, the last step of *TempoMP* is assigning tasks to the cores. The next subsection discusses how task assignment is done.

Offline optimization phase is performed in MATLAB [18]. The multiparametric programming framework is implemented in YALMIP [19] toolbox in MATLAB which relies on multi-parametric toolbox [20]. Optimization results are saved in the *power state decision module* to be accessed during the online phase of our algorithm. The *critical regions* are stored in the form of the coefficients of the linear inequalities describing them. Memory required for the *critical regions* of the MPSoC of Fig. 7 is less than 500 KB.

Algorithm 1 Power state assignment in TemPrompt

```
 1: $\mathcal{O} \Leftarrow$ types of the available cores
 2: sort $\mathcal{O}$ in the decreasing order of performance of the core types
 3: for all $\Omega$ in $\mathcal{O}$ do
 4:   $\mathcal{C}_\Omega \Leftarrow$ set of cores of type $\Omega$
 5:   sort $\mathcal{C}_\Omega$ in increasing order of the cores' $\mathcal{T}_i[k+1]$
 6: end for
 7: $P[k]$, $\mathcal{P}[k+1] \Leftarrow$ Current power of the cores
 8: $\mathcal{F} \Leftarrow \emptyset$
 9: while (Not all cores are assigned frequencies) do
10:   $\Omega \Leftarrow$ the next type in $\mathcal{O}$
11:   $\sigma_\Omega \Leftarrow$ performance requirements of core type $\Omega$
12:   $\mathcal{F} \Leftarrow \mathcal{F} \cup \sigma_\Omega$
13:   while ( ( $\mathcal{C}_\Omega \neq \emptyset$ ) and ($\mathcal{F} \neq \emptyset$) ) do
14:     $i \Leftarrow$ the next core in $\mathcal{C}_\Omega$ (with the lowest $\mathcal{T}_i[k+1]$)
15:     $p_{req} \Leftarrow$ the highest power state required from $\mathcal{F}$
16:     $p_{safe} \Leftarrow$ the highest power state for which $\mathbb{T}_o[k+1] \prec T_{Th}$
          ($\mathbb{T}_o[k+1] = \mathcal{T}[k+1] + \Phi_I(\mathcal{P}[k+1] - P[k])$)
17:     assign the lower of $p_{req}$ and $p_{safe}$ power states to core $i$
18:     update $\mathcal{P}[k+1]$ with the potential power of core $i$
19:     remove $f$ from $\mathcal{F}$
20:     remove $i$ from $\mathcal{C}_\Omega$
21:   end while
22: end while
```

B. Power State Assignment in TemPrompt

Although *TempoMP* is able to choose locally optimal power states, the results of the offline optimization must be stored and retrieved at runtime. This overhead can be high for many core systems. In order to address this issue, we propose an alternative scheduling technique,*TemPrompt*, whose power state assignment is based on a heuristic. Algorithm 1 outlines the power state assignment algorithm of *TemPrompt*.

The first step of this algorithm is sorting the core types in the decreasing order of their performance. Typically various types of cores integrated on a heterogeneous MPSoC are optimized for contrasting objectives. Usually one type is optimized for performance, while the other type is optimized for energy efficiency [21]. Therefore, various types of cores usually operate at distinct power and performance points and can be easily differentiated in terms of performance.

As our metric to quantitatively compare the performance of the core types, we use the average IPS (instructions per second) that each type achieves over the benchmarks. The algorithm first sorts the core types in the decreasing order of performance (line 2). At each stage, $\mathcal{P}[k+1]$ contains the power of potential power states of the cores (At the beginning, it is initialized to the power at the previous scheduling tick ($P[k]$)). At each stage of the algorithm, $\mathcal{F}$ keeps the performance requirements that are not still met. If a core type has not been able to meet all of its performance requirements, the remaining elements are kept in $\mathcal{F}$, so that they are addressed by the next core type (line 12).

Tempo thermal state of the core, $\mathcal{T}_i[k+1]$, is the potential temperature of core i at the end of the next scheduling tick if the power states of the cores do not change. The effect of changes in the power states of the cores are reflected by component $\Phi_{oo}(P[k+1] - P[k])$ as shown in (12). Cores with lower *Tempo* thermal state are expected to have lower temperature in the future scheduling tick and can be assigned to higher power states compared to the other cores. Therefore, the algorithm checks the instances of each type of cores in the increasing order of $\mathcal{T}_i[k+1]$.

At each step, the algorithm checks the core types in the decreasing order of their performance (line 10). The algorithm determines p_{req}, the highest power state that the current workload requires from the core type being checked (line 15). Then it checks the available cores of that type in the increasing order of their thermal state $\mathcal{T}_i[k+1]$ (line 14). For each core, it finds p_{safe}, the highest thermally safe power state of the core given the current thermal state and power settings (line 16). Core i is set to the lower of p_{req} and p_{safe} to provide just enough performance for the workload and not to exceed the temperature threshold. Then the power values are updated and then the next core is checked. If there is no core of this type left, then a new core type is examined (line 10). The algorithm finishes if the performance requirements are met or every core is assigned a power state.

The algorithm first sorts the array of the core types and the array of cores of each type. The complexity of sorting these arrays is $(\mathrm{O}(n_c \log(n_c)))$. Then the algorithm visits each core at most once $(\mathrm{O}(n_c))$. Therefore, the complexity of the power assignment algorithm in *TemPrompt* is $(\mathrm{O}(n_c \log(n_c)))$ which is typically very small ($n_c = 3$ in the heterogeneous MPSoC in the experimental results).

C. Runtime Task Assignment to the Cores

The task assignment algorithm (Algorithm 2) at each step matches the task with the highest performance requirement to the core which can provide the highest performance in the following scheduling ticks. It examines the cores based on the performance that each of them will be able to provide in the next scheduling tick. In order to do this, it starts with the highest performance core types first. Within each core type, it first examines the cores which are set to higher power states hence providing higher performance. When a core is chosen, the unassigned task with the highest performance requirement is chosen and assigned to that core. This way we make sure that the task with the highest performance requirement is provided with the highest performance core available.

performance requirement estimation module can determine power state at which each task needs to be run to be able to meet its performance requirements (deadline, throughput, etc.). Given this information from the *performance requirement estimation module* and the output of *power state decision module*, the *task assignment decision module* tries to assign the tasks to the cores such that the tasks with higher performance requirements are assigned to the cores providing higher performance. Algorithm 2 explains how this is done at each scheduling tick in a deadline-based system. This module works identically for both *TempoMP* and *TemPrompt*.

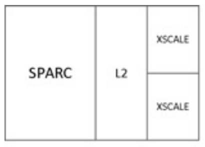
The performance requirement of a task in a deadline-based system is measured by the time remaining to its deadline. The

Algorithm 2 Task to core assignment

```
1: $\mathcal{J} \Leftarrow$ ready to run tasks
2: sort $\mathcal{J}$ in the decreasing order of the performance requirements of
   the tasks (e.g., in a deadline-based system, in increasing order of the
   deadlines)
3: $\mathcal{C} \Leftarrow$ currently available cores
4: $\mathcal{O} \Leftarrow$ types of cores in $\mathcal{C}$
5: sort $\mathcal{O}$ in the decreasing order of core type performances
6: for all $\Omega$ in $\mathcal{O}$ do
7:    $\mathcal{C}_\Omega \Leftarrow$ set of cores of type $\Omega$ in $\mathcal{C}$
8:    sort $\mathcal{C}_\Omega$ in the decreasing order of the cores' assigned power states
      (the cores with higher power states first)
9: end for
10: while $(\mathcal{C} \neq \emptyset)$ and $(\mathcal{J} \neq \emptyset)$ do
11:    $\Omega \Leftarrow$ the first element in $\mathcal{O}$ (the highest performance core type
       available)
12:    while $(\mathcal{C}_\Omega \neq \emptyset)$ and $(\mathcal{J} \neq \emptyset)$ do
13:       $j \Leftarrow$ the next element in $\mathcal{J}$ (the next ready to run task with the
          highest performance requirement)
14:       $i \Leftarrow$ the next core in $\mathcal{C}_\Omega$ (the core with the next highest power state
          among the available cores of type $\Omega$)
15:       assign task $j$ to core $i$
16:       remove $j$ from $\mathcal{J}$
17:       remove $i$ from $\mathcal{C}_\Omega$ and $\mathcal{C}$
18:    end while
19:    remove $\Omega$ from $\mathcal{O}$ and $\mathcal{C}_\Omega$ from $\mathcal{C}$
20: end while
```

shorter the time to the deadline is, the higher would be the performance requirement of the task. In other types of systems, the performance requirement of a task can be described by different measures, such as throughput requirements of each task. Our task assignment algorithm is applicable as long as the performance requirements of the tasks can be compared. The algorithm sorts the available tasks in the system in the decreasing order of their performance requirements (line 2). As an example, in a deadline-based system, the task with the highest performance requirement is the one with the earliest deadline. At each step, the task j with the highest performance requirement is chosen (line 13). Then, the core i that has the highest performance (at the highest power state) is chosen (line 14). Then task j is assigned to core i (line 15) and this is repeated until no core or no task is left. The algorithm always finishes because at each iteration it assigns a task to an available core and finishes when no core or no task is left. The algorithm always finishes, because at every iteration, it assigns a power state to one unassigned core and finishes when no unassigned core is left (the maximum number of iterations is the larger of the number of the cores and the tasks).

This algorithm sorts the core types and then sorts the available cores (lines 5, 8). The complexity of sorting these arrays is $(\mathrm{O}(n_c \log(n_c)))$. The algorithm also sorts the tasks in the decreasing order of their performance requirements (line 2) $(\mathrm{O}(n_t \log(n_t)))$. Then it pops at most n_c tasks from the list, because each core can be assigned at most one task $(\mathrm{O}(n_c))$. This way the complexity would be $(\mathrm{O}(n_t \log(n_t)))$. However, using a binary heap, no sorting is necessary for the tasks. The complexity of building the binary heap is $\mathrm{O}(n_t)$ (n_t=number of tasks) and popping each task takes $O(log(n_t))$. Because we need to pop at most n_c tasks, the complexity of the task to core assignment is $O(max(n_t, n_c log(n_t)))$. As an example, for $n_c = 3$, this value is 6 and 9 for $n_t = 4$ and $n_t = 8$ respectively. Based on the analysis of the individual algorithms, the time



(a)

Thermal Parameters	
Die Thickness	0.5 mm
Convection Resistance	7 °K/W
Convection Capacitance	100 J/°K

(b)

Processor		XScale-like	SPARC-like
Issue width		1	2
Area (mm^2)		1.4	5
Frequency	1.2 V	624	1200
Settings	1.18 V	416	1140
(MHz)	1.06 V	208	1000
Average	1.2 V	0.34	2.40
Dynamic	1.18 V	0.32	2.10
Power (W)	1.06 V	0.28	1.90

(c)

Fig. 7. Characteristics of the MPSoC. (a) Floorplan of MPSoC. (b) Thermal parameters. (c) Characteristics of the cores.

complexity of the whole algorithm (power state assignment and task assignment) is $O(max(n_r \times n_c, n_t, n_c \log(n_t))$.

V. Experimental Results

The cores used in our experiments are a low-power in-order architecture similar to the SPARC cores in UltraSPARC T1 [22], and very low power cores designed for embedded systems, similar to Intel's XScale [23]. It should be noted that these core types use the same ISA and are able to run the same binaries, but one of them is architecturally similar to SPARC, which trades off energy for performance, while the other one is architecturally similar to XScale, which trades off performance for power efficiency. Power, performance, and area characteristics of the cores are shown in Fig. 7. We assume that the MPSoC is implemented in 65 nm technology. The areas of the cores are derived from published photos of the dies after subtracting the area occupied by I/O pads, interconnection wires, interface units, L2 cache, and control logic as in [1], and scaled to 65 nm. Each L2 cache has 1 MB size, two banks, 64-byte lines, and is four-way associative. Using CACTI [24], the area and power consumption of the caches at 65 nm are estimated as 14 mm^2 and 1.7 W, respectively. The cache power consumption value includes leakage.

For performance and power data, the M5 Simulator [25] is used with Wattch [26] power model updated with 65 nm model parameters. The in-order pipelines of SPARC and Xscale are modeled by modifying M5's execution engine. We assume the same three voltage settings for the XScale and SPARC cores. For XScale, we use the existing available frequency levels (as reported in [23]), and for SPARC we set the default frequency to 1.2 GHz (as reported in [22]), and scale frequency using the 95% and 85% settings as in [7]. The overhead of switching to a new voltage/frequency is set to 50 μs. These power values are then utilized in the temperature simulations. We compute the leakage power of CPU cores based on structure areas and temperature. We compute temperature dependence using the model introduced in [27] with the same constants mentioned in this paper for 65 nm. For power overhead during voltage and frequency scaling, we use the power of the higher power state. The overhead of migration of the tasks between the cores is assumed to be 10 μs.

We use HotSpot Version 4.2 [15] for thermal modeling with a sampling interval of 100 μs to ensure sufficient accuracy. In many embedded systems such as cell phones there is no heat sink or spreader. To model this within HotSpot, we set the spreader thickness to be very thin, 0.1 mm. The heat sink is replaced by a package with thermal parameters shown in Fig. 7(b) which are within the ranges suggested by [28] and [29]. The parameters used in HotSpot are summarized in Fig. 7(b). It should be noted that this is only one example, while our techniques are general and apply to a wide range of systems with various characteristics.

The workloads in our experiments consist of integer benchmarks provided in MiBench benchmark suite [30] which include automotive/industrial, network, and telecommunications applications. Other than datasets provided in MiBench suite, we use datasets provided by [31]. We use a periodic workload model where each task has its own arrival period (τ) and deadline (d). This model allows us to compare among different techniques in terms of their ability to handle per-task performance requirements. In this periodic workload model, a new instance of each task is generated regularly at every arrival period of that task (τ) and the deadline of this task is d time units after it arrives. The deadlines are soft and a task is not dropped if its deadline is missed. To evaluate our technique under various conditions, we create moderate to intensive workloads consisting of varying number of tasks from MiBench suite. We set both deadline (d) and arrival period (τ) of each task to twice the execution time of that task at the lowest power state on the slowest core (XScale-like core). This way the tasks can potentially meet their deadlines irrespective of the core type they are assigned to. The system is preemptive and the scheduler can preempt a running task and start running another ready task. The preempted task may resume execution later.

First we compare *Tempo* with a state-of-the-art temperature predictor called band-limited predictor (BLP) [5]. It utilizes the band-limited nature of temperature frequency spectrum to predict the temperature trend. Similar to *Tempo*, the coefficients used in calculations of BLP are also calculated at design time. No training phase is required. We use the same parameters as used in [5], namely $\alpha = 0.135, m = 3$ *and* $N = 3$.

Fig. 8(a) shows the actual temperature of a core on a MPSoC along with the prediction results of *Tempo* and BLP, while Fig. 8(b) shows the trace of dynamic power applied to the same core. The sharp changes in the dynamic power are caused by power state changes happening at scheduling ticks, which are 10 ms apart. This is a slice of a longer trace of execution of the MiBench benchmarks, so the cores have high initial temperatures.

As shown in the figure, as long as the temperature and power changes are smooth, both predictors do well. However, BLP fails when the power state of the core changes significantly. For example, in Fig. 8(a), right after the first power state change at around 190 ms, BLP underestimates the temperature. This is because BLP relies exclusively on the temperature history and trend. Therefore, even if the new power state is known, BLP cannot predict the temperature before the new power state is applied and has impacted the

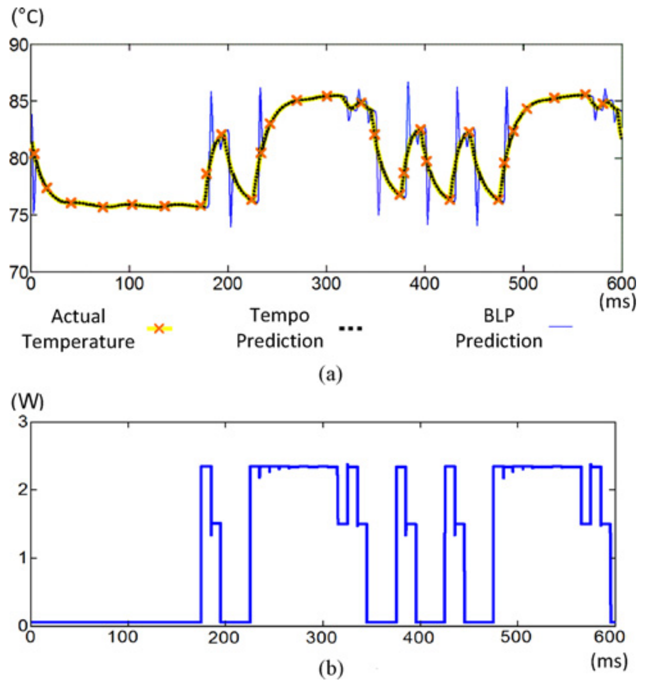


Fig. 8. Comparison of *Tempo* and BLP predictor [5]. (a) Temperature. (b) Dynamic power.

temperature trend. Moreover, as the figure shows, even after the first sample of temperature signal is observed after the change in temperature trend, BLP significantly overestimates the temperature. However, *Tempo* accurately predicts the future temperature given the next power state of the cores. As shown in this figure, the maximum prediction error of BLP can be over 5 °C, while in our experiments, the maximum temperature prediction is always less than 0.5 °C—an order of magnitude difference. BLP and any other predictors that depend only on temperature trend cannot accurately evaluate the thermal effects of scheduling decisions and power state changes. In contrast, *Tempo* can be efficiently used to evaluate alternative decisions regarding scheduling and power state changes. The results of our temperature aware scheduling techniques in *PROMETHEUS* further illustrate the efficiency of *Tempo*.

We compare *TempoMP* and *TemPrompt* with three other state-of-the-art temperature-aware scheduling techniques. The scheduling ticks are 10 ms apart for all of these scheduling techniques. The maximum safe temperature is assumed to be 90 °C that must be respected by all techniques. *PASTEMP* and *Thermal_PO* [32] are both proactive techniques which use optimization to assign thermally safe power states to the cores based on the performance requirements of the workload and also the thermal state of the system. The optimization formulation in *PASTEMP* is based on a modified dynamic thermal model called instantaneous thermal model [32], while optimization in *Thermal_PO* is based on a steady-state thermal model. None of these models can estimate temperature as accurately as *Tempo*. Therefore, to guaranty meeting thermal constraints these techniques may need to set the power states of the cores conservatively, which may result in performance loss. The third technique, *Thermal_DVFS* relies on the direct temperature readings from the thermal sensors. It switches the core to a lower power state when the core temperature

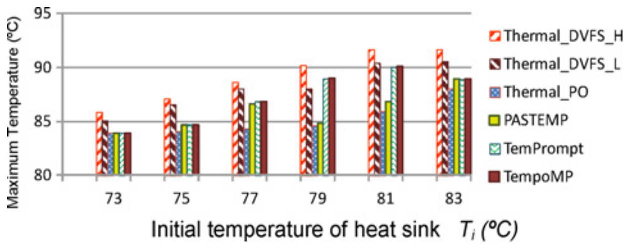


Fig. 9. Comparison of maximum temperature.

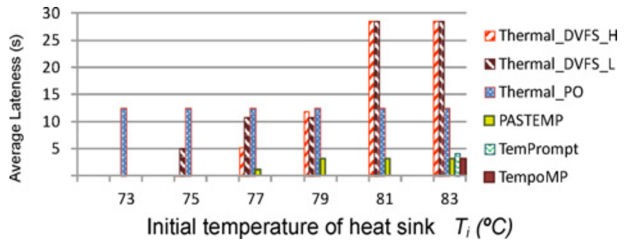


Fig. 10. Average lateness (s).

reaches a critical threshold of T_{top}. When the temperature gets below a lower threshold, T_{bottom}, the power state of the core can be switched to a higher level again. This lower threshold helps prevent oscillations between power states. We test two variations of *Thermal_DVFS* with different thresholds: *Thermal_DVFS L* with T_{top} = 85 °C and T_{bottom} = 83 °C and *Thermal_DVFS H* with T_{top} = 87 °C and T_{bottom} = 85 °C. *Thermal_DVFS* does default load balancing to create a balanced distribution of the workload across the cores.

We run same mix of MiBench benchmarks for 100 seconds. Before each run, the heat sink is pre-heated to initial temperature T_i ranging from 73 °C to 83 °C. Fig. 9 reports the maximum core temperature observed under various conditions. At low T_i, all techniques are able to meet the 90 °C temperature threshold. Only *Thermal_DVFS* violates the threshold at high T_is, because in this region the core temperature quickly reaches above the threshold before *Thermal_DVFS* can respond. *PASTEMP* and *Thermal_PO* have consistently lower maximum temperature, because their thermal models overestimate temperature, and as a result of these pessimistic temperature estimates, they tend to make more conservative scheduling decisions and use lower power states. Although this keeps the temperature lower, it results in more performance loss compared to the other techniques as shown next.

To compare how techniques address individual performance requirements of the tasks, we measure lateness of a task, which is defined as the time it takes to finish the task after its deadline is missed. Since the workload contains tasks of various types and lengths, this metric is more relevant compared to the number of deadline misses. Using lateness metric, if a task misses its deadline, as long as it is not finished, the scheduling technique still gets penalized for it. To compare how the potential processing capabilities of the MPSoC are utilized as a whole, we also look at the throughput of the MPSoC, which is the number of instructions executed per second. Fig. 10 and 11 report these two metrics. As expected, as T_i increases, so does the average lateness, while the throughput degrades.

TempoMP and *TemPrompt* miss deadlines only at the highest T_i where turning on the SPARC without violating T_{th} is

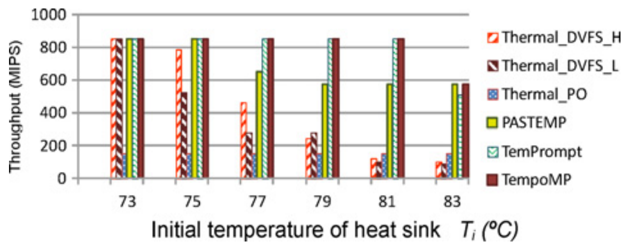


Fig. 11. Throughput (million instructions executed per second).

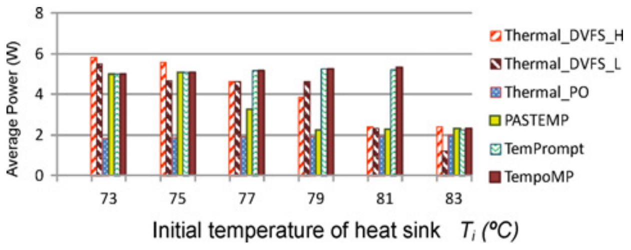


Fig. 12. Average power consumption.

not possible. In this case, no temperature aware scheduling technique can meet all the deadlines because while the performance of SPARC core is necessary to meet the deadlines, turning this core on will violate T_{th}. At the highest T_i, *TempoMP* and *PASTEMP* perform similarly and slightly better than *TemPrompt*. because they both use optimized power state assignments, while *TemPrompt* uses a heuristic approach to assign power states. Although *Thermal_PO* and *PASTEMP* are both proactive, because their temperature estimates are pessimistic compared to *Tempo*, in some cases they unnecessarily prevent cores from operating at the highest possible power states. As a result, they consistently perform worse than our techniques in terms of both lateness and throughput. *TempoMP* and *PASTEMP* improve lateness by 2.5× and throughput by 1.9× compared to average of the other techniques. To achieve this performance improvement, *TempoMP* and *TemPrompt* consume around 1.4× more power compared to average of the other techniques as shown in Fig. 12.

Fig. 12 reports average power consumption of MPSoC with DTM techniques studied. In some cases *TempoMP* and *TemPrompt* consume more power than the other techniques. These are cases where the other techniques cannot turn on large higher power cores due to their pessimistic temperature estimates; therefore, they consume lower power, but have much lower performance as well. Due to accurate estimation of thermal slack in *TempoMP* and *TemPrompt*, they are able to turn on the higher power large cores to meet the performance requirements. For example, in all cases, *Thermal_PO* consumes the least power and has the lowest performance.

We also compare these techniques using two different measures of combined energy efficiency and performance: average energy consumed per billion instructions and ELP. ELP is similar to energy-delay product (EDP) metric, but applied to the systems with deadlines. In terms of both of these metrics, *TempoMP* and *TempoMP* are on average an order of magnitude better compared to the other techniques mainly because they provide just enough performance for the workload requirements and use the larger cores—which are less power

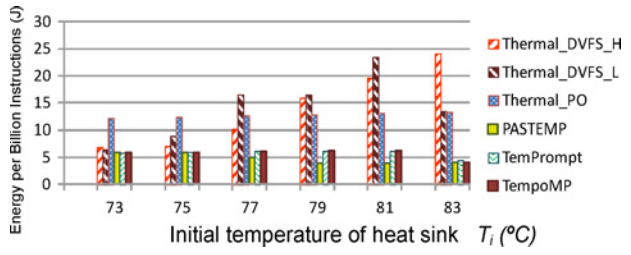


Fig. 13. Average energy per billion instructions executed.

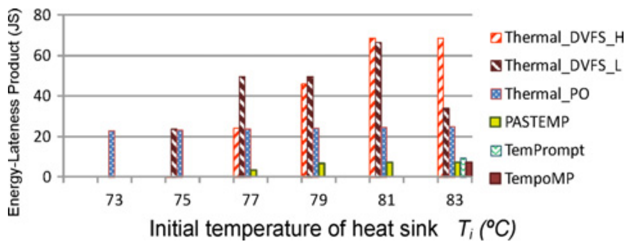


Fig. 14. Average ELP.

and energy efficient—only when their high performance is necessary. Therefore, they can better match the performance provided to the performance required. *Thermal_PO* is the least energy efficient because while it uses lower power states of the lower power cores, due to longer execution time and leakage power, it is not energy efficient.

As the experimental results show, *TempoMP* performs the best across all the techniques. It provides the best tradeoff between temperature, power, and performance. The *TemPrompt* is generally close to *TempoMP*. The main reason for superiority of these techniques over the other techniques is their accurate evaluation of the thermal impacts of future scheduling decisions. Overestimated temperature leads to conservative decisions thus not taking full advantage of thermal slack, which results in performance loss. Running the cores slower might also result in higher energy consumption due to leakage. On the other hand, underestimating future temperature can cause thermal violations and reliability issues. Taking advantage of accurate predictions of *Tempo*, *PROMETHEUS* performs better across various metrics compared to other reactive and proactive techniques. On average, *PROMETHEUS* scheduling techniques reduced the lateness of the tasks by ∼2.5× and the ELP by ∼5×.

VI. Conclusion

This paper introduced a novel temperature predictor called *Tempo* that accurately predicted the potential thermal effects of alternative power state changes in an MPSoC. *Tempo* achieved up to an order of magnitude reduction in the maximum prediction error compared to the previous predictors. We also proposed *PROMETHEUS*, a framework for proactive temperature aware scheduling of embedded workloads on heterogeneous MPSoC. Using *Tempo*, *PROMETHEUS* provided two temperature aware scheduling techniques that proactively avoided power states leading to future thermal emergencies while matching the performance provided by the heterogeneous MPSoC to the workload requirements. The first scheduling technique, *TempoMP*, integrated *Tempo* with an online multiparametric optimization method to deliver locally optimal DTM decisions to meet thermal constraints while minimizing power and maximizing performance. The second one, *TemPrompt*, used *Tempo* as a part of a heuristic algorithm that provided comparable efficiency at lower overhead. Compared to the other proactive techniques, *TempoMP* and *TempPrompt* consistently performed better in terms of performance while always meeting thermal requirements. On average, they reduced the lateness of the tasks by ~2.5× and ELP by ~5×.

REFERENCES

[1] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Processor power reduction via single-ISA heterogeneous multi-core architectures," *Comput. Architect. Lett.*, vol. 2, no. 1, pp. 5–8, Jan.–Dec. 2003.

[2] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Trans. Archit. Code Optim.*, vol. 1, no. 1, pp. 94–125, Mar. 2004.

[3] A. K. Coskun, T. Rosing, and K. C. Gross, "Utilizing predictors for efficient thermal management in multiprocessor SoCs," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1503–1516, Oct. 2009.

[4] O. Khan and S. Kundu, "Hardware/software co-design architecture for thermal management of chip multiprocessors," in *Proc. Des. Autom. Test Europe Conf.*, Apr. 2009, pp. 952–957.

[5] R. Ayoub and T. Rosing, "Predict and act: Dynamic thermal management for multi-core processors," in *Proc. Low Power Electron. Des. Int. Symp.*, 2009, pp. 99–104.

[6] F. Zanini, D. Atienza, L. Benini, and G. De Micheli, "Multicore thermal management with model predictive control," in *Proc. Circuit Theory Des. Eur. Conf.*, Aug. 2009, pp. 711–714.

[7] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Proc. Microarchitect. Annu. IEEE/ACM Int. Symp.*, 2006, pp. 359–370.

[8] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, L. Benini, and G. De Micheli, "Temperature control of high-performance multi-core platforms using convex optimization," in *Proc. Conf. Des. Autom. Test Eur.*, 2008, pp. 110–115.

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

[10] T. Chantem, R. Dick, and X. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," in *Proc. Des. Autom. Test Eur.*, 2008, pp. 288–293.

[11] S. Sharifi, A. Coskun, and T. Rosing, "Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor SoCs," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2010, pp. 873–878.

[12] S. Sharifi and T. Rosing, "Accurate direct and indirect on-chip temperature sensing for efficient dynamic thermal management," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 29, no. 10, pp. 1586–1599, Oct. 2010.

[13] Y. Liu, R. Dick, L. Shang, and H. Yang, "Accurate temperature-dependent integrated circuit leakage power estimation is easy," in *Proc. Des. Autom. Test Eur.*, Apr. 2007, pp. 1–6.

[14] G. Golub and C. Van Loan, *Matrix Computations*, 2nd ed. Baltimore, MD, USA: The Johns Hopkins Univ. Press, 1989.

[15] *Hotspot Temperature Modeling Tool* (2010) [Online]. Available: http://lava.cs.virginia. edu/HotSpot/

[16] R. Ayoub *et al.*, "OS-level power minimization under tight performance constraints in general purpose systems," in *Proc. Low Power Electron. Des. Int. Symp.*, Aug. 2011, pp. 321–326.

[17] D. Narciso *et al.*, "A framework for multi-parametric programming and control: An overview," in *Proc. IEEE Int. Eng. Manage. Conf. IEMC Eur.*, 2008, pp. 1–5.

[18] *MATLAB*, version 7.10.0 (R2010a), The MathWorks, Inc., 2010.

[19] *Yalmip*. (2011) [Online]. Available: http://users.isy.liu.se/johanl/yalmip/.

[20] *Multi-Parametric Toolbox (MPT)*. (2011) [Online]. Available: http://control.ee.ethz.ch/mpt/.

[21] K. C. Hyun-Duk Cho and TaehoonKim, "Benefits of the big.LITTLE Architecture," 2012, Samsung White Paper.

[22] A. S. Leon, K. W. Tam, J. L. Shin, D. Weisner, and F. Schumacher, "A power-efficient high-throughput 32-thread SPARC processor," *IEEE J. Solid-State Circuits*, vol. 42, no. 1, pp. 7–16, Jan. 2007.

[23] Intel. (2011). *Intel PXA270 Processor, Electrical, Mechanical and Thermal Specification Data Sheet* [Online]. Available: http://www.intel.com

[24] HP. (2011). *CACTI* [Online]. Available: http://www.hpl.hp.com/research/cacti

[25] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, Jul. 2006.

[26] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proc. Int. Symp. Comput. Architect.*, Jun. 2000, pp. 83–94.

[27] S. Heo, K. Barr, and K. Asanović, "Reducing power density through activity migration," in *Proc. Int. Symp. Low Power Electron. Des.*, 2003, pp. 217–222.

[28] JEDEC Solid State Technology Association, JEDEC Standard 51-2A, Integrated Circuit Thermal Test Method Environmental Conditions-Natural Convection (Still Air).

[29] F. Mohammadi and M. Marami, "Dynamic compact thermal model of a package," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2008, pp. 2869–2872.

[30] M. R. Guthaus *et al.*, "Mibench: A free, commercially representative embedded benchmark suite," in *Proc. IEEE Int. Workshop Workload Characterization*, Dec. 2001, pp. 3–14.

[31] G. Fursin *et al.*, "MiDataSets: Creating the conditions for a more realistic evaluation of iterative optimization," in *Proc. Int. Conf. High Performance Embedded Architect. Compilers*, 2007, pp. 245–260.

[32] S. Sharifi and T. Rosing, "Package-aware scheduling of embedded workloads for temperature and energy management on heterogeneous MPSoCs," in *Proc. Int. Conf. Comput. Des.*, 2010, pp. 541–546.

Shervin Sharifi (M'13) received the B.S. degree from the Sharif University of Technology, Tehran, Iran, the M.S. degree from the University of Tehran, Tehran, Iran, and the Ph.D. degree in computer engineering from the University of California, San Diego (UCSD), CA, USA, in 2011

Prior to joining UCSD, he was a Digital System Designer and Electronic Design Automation Software Engineer. He is currently with the System-on-Chip Architecture Group, Qualcomm Technologies, Inc., San Diego, His current research interests include system-on-chip (SoC) architecture, embedded systems, and power and temperature aware system design.

Dilip Krishnaswamy (SM'12) received the Ph.D. degree in electrical engineering from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 1997.

He is currently a Senior Staff Researcher at the Office of the Chief Scientist, Qualcomm Research Center, San Diego, CA, USA. He was a Platform Architect at Intel, Folsom, CA, USA, and a Lead Architect at the PXA800F, Intel's first cellular heterogeneous multiprocessor SoC platform. His current research interests include multiprocessing systems, wireless distributed computing, distributed analytics, m2m technologies and systems, nanobiological systems, nanointerconnects, distributed cooperative processing, and parallel wireless processing.

Dr. Krishnaswamy is an Associate Editor-in-Chief of the IEEE Wireless Communications Magazine. He chairs the IEEE COMSOC Emerging Technical Subcommittee on Applications of Nanotechnology in Communications.

Tajana Šimunić Rosing (M'01) received the Ph.D. degree from Stanford University, Stanford, CA, USA, in 2001, concurrently with finishing the Masters degree in engineering management. Her Ph.D. topic was dynamic management of power consumption.

She is currently an Associate Professor at the Computer Science Department, University of California, San Diego (UCSD), CA, USA. She is currently heading the effort in energy efficient data centers as a part of MuSyC Center, Berkeley, CA, USA. Prior to this, she was a full-time Researcher at HP Labs, Palo Alto, CA, USA, while working part time at Stanford University. Prior to pursuing the Ph.D. degree, she was a Senior Design Engineer at Altera Corporation, San Jose, CA, USA. Her current research interests include energy efficient computing, and embedded and wireless systems.

Dr. Rosing is currently an Associate Editor of the IEEE TRANSACTIONS ON MOBILE COMPUTING.