Temperature Aware Dynamic Workload Scheduling in Multisocket CPU Servers

Raid Ayoub, *Student Member, IEEE,* Krishnam Indukuri, *Member, IEEE,* and Tajana Simunic Rosing, *Member, IEEE*

***Abstract*—In this paper, we propose a multitier approach for significantly lowering the cooling costs associated with fan subsystems without compromising the system performance. Our technique manages the fan speed by intelligently allocating the workload at the core level as well as at the CPU socket level. At the core level we propose a proactive dynamic thermal management scheme. We introduce a new predictor that utilizes the band-limited property of the temperature frequency spectrum. A big advantage of our predictor is that it does not require the costly training phase and still maintains high accuracy. At the socket level, we use control theoretic approach to develop a stable scheduler that reduces the cooling costs further by providing a better thermal distribution. Our thermal management scheme incorporates runtime workload characterization to perform efficient thermally aware scheduling. The experimental results show that our approach delivers an average cooling energy savings of 80% compared to the state of the art techniques. The reported results also show that our formal technique maintains stability while heuristic solutions fail in this aspect.**

***Index Terms*—Fan control, multiple cores, multiple CPU sockets, multitier thermal management, state-space control, temperature prediction.**

I. Introduction

MODERN servers are commonly equipped with multiple CPU sockets to cope with the increasing demand of computationally intensive applications [1], [2]. To further increase the computational power, an additional layer of parallel processing is implemented within the CPU socket where each socket is a chip multiprocessors (CMP, e.g., Intel Xeon quad core processor). However, this complex level of integration coupled with high performance of the processors leads to higher power densities [3]. The high power density causes thermal hot-spots in the system that have substantial effect on reliability and leakage power [4]. It also degrades performance since interconnect delays increase with temperature [5]. Dissipating the excess heat is one of the biggest challenges as it requires a complex and energy hungry cooling subsystem.

The cooling subsystems in high-end servers are designed based on the concept of forced convection where controlling the rate of air flow improves the heat transfer between the heat sink and the ambient to meet the desired value. The air flow is normally generated using a fan subsystem that focuses the air toward the CPU heat sink. To control the fan speed, closed loop controllers are normally used where CPU thermal sensors provide feedback signal to the controller. Fan-based cooling subsystem increase the air-flow rate to match to a corresponding rise in temperature. However, the challenge is the substantial increase in cooling power due to the cubic relationship between fan speed and its power [6]. The fan system in high-end servers consumes as much as 80 W in 1U rack servers [6] and 240 W or more in 2U rack servers [1]. The results in [7] show that the fan power can reach up to 51% of the overall server power budget. Moreover, the increase in fan speed introduces large noise levels in the system. The acoustic noise levels increase by 10 dB as air-flow rate increases by 50% [8]. Such increase in noise level not only leads to uncomfortable working environment but also causes vibrations that may impact reliability. Minimizing rate of air flow to just what is needed for cooling is essential to deliver energy efficiency at minimal acoustic noise level.

Current operating systems employ dynamic load balancing (DLB) to enhance the utilization of the system resources. The DLB performs thread migration to lower the difference in task queue lengths of the individual computational units [9]. Nevertheless, thermal hot spots my still occur in the CPUs since DLB does not consider temperature in allocating the workload. When the number of running threads is less than the number of physical cores, the DLB does not initiate any migrations since the workload is balanced from performance point of view. Such scenarios can result in hot-spots as a portion of the cores could be highly active while the others are idle.

To manage the high temperature within a single CPU socket, a number of dynamic thermal management (DTM) techniques have been proposed. Reactive techniques, proposed in the literature, manage the temperature upon reaching a critical threshold. Employing these techniques come at a high price of performance overhead. A few predictive thermal migration techniques have been proposed recently that can predict the occurrence of thermal emergencies ahead of time [10]–[12]. Although temperature prediction is fairly accurate, a training phase is required that impacts the performance and prediction opportunities. Besides, prior techniques are limited to single socket CPU designs and cooling dynamics are not modeled with sufficient accuracy. We identify these as a major drawbacks that motivates us to propose a multitier proactive work-

Manuscript received January 25, 2011; accepted March 21, 2011. Date of current version August 19, 2011. This work was supported by the National Science Foundation (NSF) Project GreenLight, under Grant 0821155, the NSF SHF, under Grant 0916127, the NSF ERC CIAM, the NSF Variability, the NSF Flash Gorden, CNS, the NSF IRNC, Translight/Starlight, Oracle, Google, Microsoft, MuSyC, UC Micro, under Grant 08-039, and Cisco. This paper was recommended by Associate Editor Y. Xie.

The authors are with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093 USA (e-mail: rayoub@cs.ucd.edu; kindukur@ucsd.edu; tajana@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.2011.2153852

load scheduling algorithm to significantly lower the *cooling costs* (power consumed for cooling) and performance overhead due to thermal emergencies while ensuring a stable solution.

In general, at any given speed setting the fan can dissipate only a limited amount of heat from a CPU. Increasing the speed requires exponential increase in fan power. This indicates that temperature solutions that act only within a socket are not sufficient to minimize cooling energy since some sockets may generate much more heat than others, which requires a better heat balance between them. This motivates us to develop a multitier algorithm that schedules the workload at the core and socket levels to minimize cooling energy and the occurrence of thermal emergencies.

We schedule the workload between CPU sockets in a way that mitigates hot spots across them and reduces cooling energy. We developed a control theoretic framework for the socket level scheduling that guarantees the desired objectives, in terms energy savings and stability, are met.

We add core level workload management to reduce the hot spots across the cores and improve cooling savings within a given socket. We propose a novel proactive workload management scheme to distribute the workload between cores in a thermally sensitive manner. In order to predict temperature accurately at negligible cost, we present a new temperature predictor, called band-limited predictor (BLP) that is based on the band limited property of the temperature frequency spectrum [13]–[15]. The important feature of our predictor is that the prediction coefficients can be computed at the design stage that makes it workload independent. The analysis that we provide shows that this predictor is not only accurate but also very cost efficient.

The rest of the paper is organized as follows. Related work and the overview of our multitier algorithm are discussed in Sections II and III, respectively. Sections IV and V describe our core level and socket level schedulers. In Section VI, the evaluation methodology and the experimental results are discussed. The conclusion is provided in Section VII.

II. Related Work

In recent years, a number of core level thermal management techniques have been suggested. They can be broadly classified into two categories: reactive and proactive management techniques. The research in [4] proposes two reactive DTM techniques that manage aggressive heat. The first technique is based on dynamic voltage–frequency scaling while the other use pipeline throttling to remove the excess heat. Recently, DTM has become an integral part of actual processor designs. For example, the Xeon processor employs clock gating and dynamic voltage scaling to manage thermal emergencies. The research in [9] and [16] proposes activity migration to manage the excess temperature by moving the computations across duplicated units. The common drawback with these approaches are the performance overhead and poor thermal distribution since they react only upon reaching thermal emergencies.

To overcome the problems with reactive techniques, a class of proactive thermal management techniques have been suggested that try to avoid the overheating while the processor is still running below the temperature threshold. The authors in [10] propose the ARMA model that is based on the serial autocorrelation in the temperature time series data. The model is updated dynamically to adapt to possible workload changes. Although the ARMA model is fairly accurate, it requires a training phase that could impact the performance, and the predictor could miss some of the prediction opportunities during training. The authors in [11] and [12] suggest proactive thermal management techniques that utilize regression-based thermal predictors. However, these techniques also require runtime adaptation. Moreover, these techniques do not model cooling dynamics with sufficient accuracy that we identify as another major limitation.

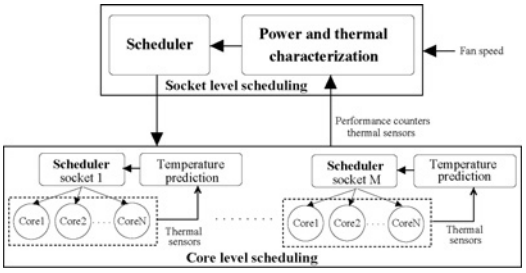

Fig. 1. Overview of multitier scheduling.

Recently, a few fan control algorithms have been suggested that operate based on closed loop control schemes to optimize the fan controller [17], [18]. The research in [17] suggests an optimal fan speed mechanism for the blade servers. The optimal speed is determined through convex optimization. The work in [19] suggests a fan control mechanism that also consider the leakage power. The research in [20] and [21] address the challenges of thermal management and cooling in data centers. In [6] and [22], methodologies are proposed for modeling the convective thermal resistance between the heat sink and ambient temperature as a function of the air-flow rate, which we use here.

In this paper, we address the limitations of the previously suggested thermal management techniques in single machines. First, the scope of the techniques that manage temperature within single machines is limited to single sockets, hence they cannot minimize hot-spots between sockets. Second, the temperature management within the single socket come with drawbacks that require improvements. The other important limitation is the lack of a realistic model of the cooling subsystem. Our study focuses on mitigating the limitation in the prior work to reduce operational costs and to enhance performance. The main contributions of our study are summarized as follows.

1) We design a new cooling aware multitier dynamic workload scheduling technique within a control theoretic framework to deliver high energy savings and ensure stability.
2) We introduce a novel temperature predictor that is accurate and workload independent and it is designed based on the band limited property of the temperature.
3) We present a thorough evaluation and discussion of the proposed techniques that results in a substantial cooling costs savings of 80%.

III. Multitier Thermal Management Overview

In general, the heat removed from the CPU by the fan is proportional to the fan speed. However, removing more heat requires exponential increase in cooling power. This means it is better to have the heat dissipation balanced between the CPUs to have a more uniform fan speed distribution and thus reduce the cooling energy. The other possible way to reduce cooling energy is to lower the average combined fan speeds. This can be achieved by eliminating some of the hot spots and maximizing the use of fan capacity. Implementing these optimizations require intelligent job allocation between sockets. We can also reduce temperature by decreasing the power density on the die through migrating the workload from hot cores to cooler cores. This class of optimizations requires core level scheduling. This indicates that utilizing both core level and socket levels is necessary in order to maximize cooling savings.

These concepts motivate us to develop a multilevel thermal management (MTTM) technique where temperature is managed at socket as well as at core level. Fig. 1 depicts the operational framework of our proposed approach.

A. Socket Level

The scheduler at the socket level manages the jobs between sockets. It takes temperature, performance, and fan speed information as an input. They are collected every during scheduling period. Cooling savings can be achieved by balancing fan speed or lowering the average combined fan speed. The techniques that achieve these two objectives are as follows.

1) *Spreading*: This method focus one the cases when power is significantly unbalanced between sockets. It tries to balance the thermal hot spots across the sockets to generate a more uniform distribution in fan speed.
2) *Consolidation*: This technique is complementary to spreading where it optimize the situations where the sockets have balanced power distributions. It schedules the workload in a way that eliminates subset of the hot spots, an issue that reduce the average of combined fan speeds and deliver cooling savings.

We design our scheduler within a control theoretic framework to ensure stability and to avoid exhaustive tuning as in the case of heuristic solutions. The scheduling period is on the order of seconds that incurs negligible overhead to the performance.

B. Core Level

Core level is complementary to the socket scheduling where it reduces the temperature within each individual socket. The core level scheduler employs proactive thermal management to minimize the temperature. To predict the temperature, we use our novel temperature predictor that ensures accuracy without the need of runtime adaptation. The scheduler calculates the predicted temperature of the individual cores and migrates the jobs from the hot cores to those that are predicted to be the coldest. This strategy helps reduce the occurrence of hot spots by providing a better thermal balance between cores. The core level algorithm is invoked during the operating system (OS) scheduling periods that is in the order of milliseconds. This core level scheduler should not impact the stability of the system since it reaches steady state within a small fraction (several 10s of milliseconds) of the high-level scheduler interval (several seconds). We show that the overhead of our technique is trivial due to the low time overhead of both predictions and migrations compared to the time scale of temperature changes. In the subsequent sections, we discuss both core and socket level algorithms in more details.

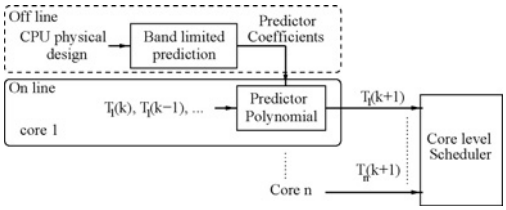

Fig. 2. Overview of core level scheduling.

IV. Core Level Scheduling

The aim of the core level scheduler is to distribute the workload across the cores in a single socket in a thermally sensitive manner. The primary objectives are to minimize the cooling costs and to reduce the frequency of hot spots. We predict the temperature of the individual cores in order to capture thermal emergencies in advance and to prevent them with a better workload assignment.

Our proactive algorithm performs thermal aware scheduling by migrating the jobs from the hot cores to cold ones and also by scheduling the incoming jobs in a thermally aware manner. Fig. 2 shows the overview of the core level scheduling. To predict the temperature we use a fundamentally different way compared to the predictors that have been suggested in the literature. Those predictors use time domain methods that make them workload dependent and are prone to run time adaptations that hurt performance. Our solution is to use frequency domain since the temperature's frequency response is a function of the chip's physical characteristics that makes it workload independent. Thus, the predictor parameters can be estimated accurately at the design stage. The other big advantage of our technique is that it is highly accurate and the prediction can be done at almost no cost. The details of our prediction method are discussed below. The thermal management is performed at OS scheduling ticks with period in the order of few milliseconds. The core level migration overhead is only a few microseconds that introduce lesser cost than putting the processor in a low power mode for cooling [9], [23]. The migration cost comes primarily from migrating the thread state and the respective warm up time of the private cache [23].

A. Temperature Prediction

Our new temperature predictor is used as an input to the core level scheduler as depicted in Fig. 2. The basic idea of the temperature predictor is that the band-limited signals can be predicted from the previous samples using prediction coefficients that are *independent* of the particular signal or autocorrelation function [13]–[15]. The prediction coefficients are function of the signal bandwidth only. Before going into the details of our BLP, we show that the temperature frequency spectrum is band-limited in nature.

Using the well-known duality between heat transfer and electrical current, the temperature can be modeled as an *RC* network [3]. The temperature of the individual die components (e.g., cores, L2 cache) are modeled as a simple *low-pass RC* filter with horizontal thermal resistances connecting the adjacent units. The bandwidth of temperature frequency spectrum can be computed using standard *RC* network analysis or using computer-aided design tools, e.g., HSPICE. For high-end CPUs, we can neglect the effect of the horizontal thermal resistances to simplify the bandwidth computations since their values are much higher than the vertical ones [16]. As a result, the temperature frequency spectrum of the individual components can be modeled as follows:

$$\frac{T(w)}{T(0)} = \frac{1}{\sqrt{1+(w\tau_c)^2}} \tag{1}$$

where $T(w)$ represents the temperature value as a function of the angular frequency w, and τ_c is the core temperature time constant that equals RC. Given that the temperature is a low-pass filter, it satisfies the band-limited condition. The author in [15] shows that band limited signals can be predicted using the linear formula as follows:

$$x(t) = \sum_{n=1}^{N} a_n x(t_n) \tag{2}$$

where a_n are the prediction coefficients, t_n represents the nth time sample and N is the total number of samples. For the case of uniform sampling, the value of t_n can be written as $t_n = t - nd_s$ where d_s is the sampling period. For this predictor to be applicable, the error needs to be bounded and sufficiently small. The absolute error is expressed as $\epsilon = |x(t) - \sum_{n=1}^{N} a_n x(t_n)|$. Using Paley–Wiener theorem and Schwarz formula, a bound on the error ϵ^2 can be found [15] as follows:

$$\epsilon^2 \leq \{\int_{-W}^{W} |X(f)|^2 df\}\{\int_{-W}^{W} |ds(f)|^2 df\} \tag{3}$$

where $ds(f) = e^{i2\pi ft} - \sum_{n=1}^{N} a_n e^{i2\pi f t_n}$, f is the frequency and W is the frequency bandwidth in Hertz. The first part of this equation represents the signal energy while the second part is the amount of prediction error. Equation (3) can be rewritten as follows:

$$\epsilon^2 \leq \|x\|^2 \cdot \epsilon_I \tag{4}$$

where ϵ_I is the error component while $\|x\|$ is the signal energy. The error integral represents an N-dimensional function $E(a_1, a_2, \ldots, a_N)$ of prediction coefficients. The BLP coefficients can be obtained by minimizing ϵ_I. To obtain the optimal coefficients, we use the method of *eigenvector optimization* [15]. The work in [24] shows that the error integral part ϵ_I can be rewritten as follows:

$$\epsilon_I = v^T \Omega v \tag{5}$$

where v is a vector that have $N+1$ elements. To satisfy this equation, it is imperative for the vector v to have its first entry equals to 1 ($v_1 = 1$). The matrix Ω is expressed as follows:

$$\Omega = \begin{pmatrix} 1 & \mathbf{b}^T \\ \mathbf{b} & D \end{pmatrix}$$

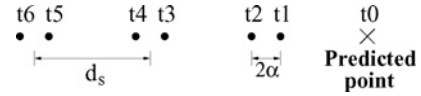

Fig. 3. Position of interlaced sampling points.

where $\mathbf{b}$ is a vector of N components and D is a matrix of $N \times N$. The vector $\mathbf{b}$ and the matrix D can be constructed from the results of applying the standard method of minimization for $E(a_1, a_2, \ldots, a_N)$ [14], to the error that gives the system of equations as follows:

$$\sum_{n=1}^{N} a_n sinc(2W(t_j - t_n)) = sinc(2W(t_0 - t_j)) \tag{6}$$

where $j = 1, \ldots, N$ and $sinc(t) = sin(\pi t)/(\pi t)$. The matrix D equals the system matrix in (6) and the vector $\mathbf{b}$ equals the vector that is in the right-hand side of (6). As a result, the matrix Ω can be expressed as follows:

$$\Omega = \begin{pmatrix} s(t_0-t_0) & s(t_1-t_0) & s(t_2-t_0) & \ldots & s(t_N-t_0) \\ s(t_0-t_1) & s(t_1-t_1) & s(t_2-t_1) & \ldots & s(t_N-t_1) \\ s(t_0-t_2) & s(t_1-t_2) & s(t_2-t_2) & \ldots & s(t_N-t_2) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ s(t_0-t_N) & s(t_1-t_N) & s(t_2-t_N) & \ldots & s(t_N-t_N) \end{pmatrix}$$

where $s(t) = sinc(2Wt)$. The matrix Ω is symmetric since $sinc(t)$ is an even function. In addition, this matrix is positive definite that makes all the eigenvalues positive. The optimal prediction coefficients that minimize the prediction error ϵ_I can be obtained by minimizing the value of $v^T \Omega v$ across all the possible vectors with $v_1 = 1$. For the set of the normalized eigenvectors with length 1, the minimum value of $v^T \Omega v$ is determined by the smallest eigenvalue of matrix Ω. In other words, we need to find the eigenvector that is associated with the smallest eigenvalue, $\lambda_{\min}$, then normalize this eigenvector to make the first entry equals to 1. Lets assumes that V is the eigenvector that is associated with the smallest eigenvalue with $\|V\| = 1$. For this selected eigenvector, the value of $V^T \Omega V$ is equal to $\lambda_{\min}$. The normalized eigenvector, V_{norm}, can be obtained simply by dividing the eigenvector by the vector's first element, $V_{\text{norm}} = \frac{V}{V_1}$. The important result is that we can extract the set of prediction coefficients, $\{a_1, a_2, \ldots, a_N\}$, directly from V_{norm} [15] as follows:

$$a_i = V_{norm_{i+1}}. \tag{7}$$

The significance of (7) is that the optimal prediction coefficients depend only on the signal bandwidth, W. For the case of uniform sampling, the upper bound for prediction distance, d_p, can be obtained using Nyquist condition, $2d_pW < 1$. The temperature spectral bandwidth depends on the value of τ_c as shown in (1). Extracting τ_c can be simply accomplished at the design time using die layout and thermal package parameters [3].

The temperature typically changes slowly as a function of its thermal time constant. As a result, extending the prediction window allows for better thermal management as more thermal emergencies could be captured and prevented ahead of time. The prediction window can be extended by employing nonuniform sampling. The theoretical work in [25] shows that if a signal is sampled at $1/m$ times the Nyquist rate, but in each sampling interval not one but m samples are used, the signal

Algorithm 1 Core Level Thermal Management

1: Calculate T^p of all cores then find T^p_{max} and T^p_{min}
2: **while** (unmarked cores > 1 **AND** $T^p_{max} > T^m_c$ **AND** $T^p_{max} - T^p_{min} > \delta_m$) **do**
3: $core_{hot} \Leftarrow$ predicted hottest core
4: $core_{cold} \Leftarrow$ predicted coolest core
5: **if** ($core_{cold}$ is idle) **then**
6: Migrate the workload of $core_{hot}$ to $core_{cold}$ core then mark them
7: **else**
8: Swap the workload of $core_{hot}$ and $core_{cold}$ then mark them
9: **end if**
10: Find new T^p_{max} and T^p_{min} from the remaining set of unmarked cores
11: **end while**
12: Assign incoming jobs to the coolest cores available

can be reconstructed completely. In [15], the author show that this concept can be extended to increase the prediction window. To apply this concept, the samples need to be placed in interlaced pattern. For $m = 2$, the placement of samples should follow a pattern of two close samples that is followed by a uniform delay, then two more close samples are taken, and so on. Fig. 3 shows the locations of interlaced sampling points. The theoretical proof is given in [15]. The prediction distance is computed as $d_p = d_s - 2\alpha(m - 1)$ where α is the half distance between the two close samplings.

In summary, to predict the temperature, we first compute the coefficient factors at the design time using (7). At run time, we collect the temperature samples and apply the simple polynomial given in (2) to predict the temperature, as shown in Fig. 2. The overhead of our prediction is negligible since computing the prediction polynomial can be achieved in a few CPU cycles.

B. Core Level Proactive Thermal Management (CPTM)

The details of the algorithm are described in Algorithm 1. This algorithm performs thermal aware scheduling by migrating the jobs from the hot cores to cold ones and also by scheduling the incoming jobs in a thermally aware manner. To initiate the migration process, the predicted temperatures, T^p, of the hot cores must surpass the migration threshold, T^m_c, and there has to be a sufficient difference in the predicted temperature, δ_m, between the hot and cold cores (Steps 2–6). If the cold core is executing, then we swap the workload of the hot and the cold cores (Step 8). When assigning the incoming jobs, we use the result of prediction to find available cold cores (Step 12).

V. Socket Level Scheduling

We introduce our socket level scheduling to provide a better thermal distribution across different CPU packages. This provides additional savings on top of those obtained via core level scheduling. Fig. 4 shows the overview scheduling at this level. It is composed of two primary stages: controller and scheduler. The controller determines how much power needs to be moved to/from sockets to minimize the cooling energy. The scheduler takes the inputs from the controller, thermal sensors and performance counters and tries to assign the workload based on the controller decisions. The scheduler communicates with a cooling savings estimator to quantify the savings of each decision and thus avoid ineffective scheduling events. To better understand this design we start with description of thermal and cooling models we used.

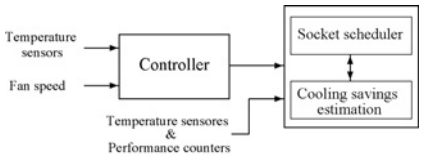

Fig. 4. Overview of socket level scheduling.

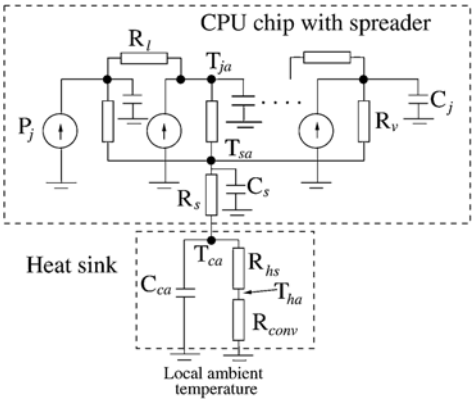

Fig. 5. Single socket thermal model.

A. Thermal and Cooling Modeling

Thermal modeling of the individual sockets can be done using an *RC* network similar to what is used in HotSpot [26]. Fig. 5 shows the thermal model of the CPU chip with a thermal package. The thermal model of the CPU chip includes the die and the heat spreader. R_v is the die component's vertical thermal resistance including the thermal interface resistance. The lateral heat flow between the die components is modeled using a lateral thermal resistance R_l. However, the impact of the lateral heat flow can be ignored due to the high ratio of core area to die thickness in high-performance processors [16]. C_j is the thermal capacitance of the components in the die and P_j corresponds to the power dissipated by the individual components. For the spreader, R_s and C_s refer to the thermal resistance and capacitance of the spreader, respectively. The heat spreader is simplified to a single node because it behaves as an isothermal layer due to its high thermal conductivity. The time constant of the core is about 60 times smaller than that of the heat spreader [27].

In state-of-the-art servers, CPU sockets are equipped with fans for cooling. For example, the Intel s5400sf server has two sockets where each socket has two sets of fans [2] that blow air toward its heat sink. The heat flow between the CPU case to ambient can be modeled by a combination of conduction and convection heat transfers, [26]. The heat sink is assumed to be an isothermal layer due to its high thermal conductance [26]. $R_{\rm hs}$ represents the thermal conductive resistance of the heat sink. The convective part is modeled by a convective resistance, $R_{\rm conv}$, connected in series with $R_{\rm hs}$ where their sum represents the case to ambient resistance, R_{ca}. R_{ca} is connected in parallel with the thermal capacitance of the heat sink, $C_{\rm hs}$, to form a simple *RC* circuit that has a time constant that is in the orders of magnitude larger than the time constant of the heat spreader [27]. The reference temperature, *local ambient temperature*, measured inside of the server enclosure is normally higher than the room's ambient temperature by

TABLE I
TEMPERATURE PROFILE OF A CPU WITH HEAT SINK

Workload	$T_{ja}^{\max}$	T_{sa}	T_{ca}	T_{ha}
{9 W, 9 W, 9 W, 9 W}	71.6 °C	54.0–54.8 °C	52.6–53.3 °C	50.1–50.8 °C
{13 W, 9 W, 7 W, 7 W}	79.3 °C	53.9–54.8 °C	52.5–53.2 °C	50.2–50.9 °C

20 °C typically [28]. We call the temperature between the CPU case to the local ambient as T_{ca}.

In [6] and [22], it is shown that the value of R_{conv} changes with the air-flow rate. Unfortunately, HotSpot uses a fixed R_{conv} since it assumes a fixed fan speed, which does not represent real systems. Using the results in [6] and [22], the value of R_{conv} can be computed as follows:

$$R_{\text{conv}} \propto \frac{1}{AV^{\alpha}} \tag{8}$$

where A is the heat sink effective area, V is the air-flow rate and α is a factor with a range of 0.9–1.0 for heat sinks in high-end servers. To estimate the cooling costs we use the results from [6] to relate the fan speed, F, with the air-flow rate as $V \propto F$. The cooling costs for changing the air-flow rate from V_1 to V_2 can be computed [6], [19] as follows:

$$\frac{P_{V2}}{P_{V1}} = \left(\frac{V_2}{V_1}\right)^3 \tag{9}$$

where P_{V1} and P_{V2} represent the fan's power dissipation at V_1 and V_2, respectively. Next, we calculate the amount of fan power that is required to reduce T_{ca} from T_{ca_1} to T_{ca_2}. For a given CPU power, and using (8), we can write $\frac{V_2}{V_1} = (\frac{T_{ca_1}}{T_{ca_2}})^{\frac{1}{\alpha}}$. Using this result and (9), we get

$$\frac{P_{V2}}{P_{V1}} = \left(\frac{T_{ca_1}}{T_{ca_2}}\right)^{\frac{3}{\alpha}}. \tag{10}$$

This shows that optimizing the fan speed is crucial for power savings since reducing T_{ca} requires an increase in the fan power on the order of $\frac{3}{\alpha}$.

Verification of thermal and cooling models: We start with showing that the value of T_{ca} is a function of the total power dissipated in the CPU socket rather than specific temperature distribution of the cores. We illustrate our ideas using HotSpot simulator that we extend to include the dependency of R_{conv} on the air-flow rate. We assume a 4 core CPU socket with a floor plan and thermal package similar to a quad core Intel Xeon [22], an air flow of 20 cubic feet per minute (CFM) and local ambient temperature inside server enclosure of 42 °C. We run two cases as shown in Table I. In the first case, we simulate execute four threads where each thread dissipates 9 W of total power. For the second case, we run four threads that accumulate similar total power to the first case but with large variation in the power distribution {13 W, 9 W, 7 W, 7 W}. The results show that T_{ca} of the two cases stay almost the same despite the large difference between their peak temperatures, $T_{ja}^{\max}$. The results also show similar behavior for the heat spreader temperature, T_{sa}, and temperature between heat sink surface to local ambient, T_{ha}.

The other important feature that we explore is the correlation between $T_{ja}^{\max}$ and T_{ca}. Fig. 6 shows how $T_{ja}^{\max}$ and T_{ca} changes with fan speed and executing various workloads that have different dynamic power distribution. Leakage power is calculated using a model that accounts for

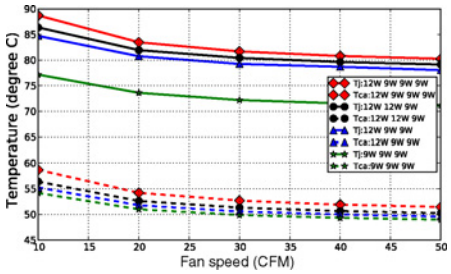

Fig. 6. Impact of fan speed on core and case to ambient temperature.

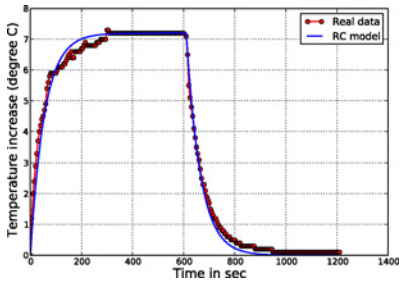

Fig. 7. T_{ha} transient behavior (referenced to idle temperature).

temperature effect on leakage that we give later in Fig. 11. It can be noticed that $T_{ja}^{\max}$ changes at a rate similar to T_{ca}. This is because T_{ca} is connected in series with the junction to case temperature. The value of $T_{ja}^{\max}$ in the case of {12 W, 9 W, 9 W, 9 W} is higher than the one in {12 W, 12 W, 9 W} despite the fact that the later has two hot threads at 12 W while the former has only one. The reason is that the later workload has lower total power that is manifested by its lower T_{ca}. Increasing the fan speed reduces the value of T_{ca} due to the reduction in R_{conv}. These results indicate that $T_{\max}$ can be used to control $T_{ja}^{\max}$ and the cooling rate.

We extend the evaluation of the cooling model by running experiments on a real 45 nm Intel Quad Core dual socket Xeon E5440 machine. The aim of the first experiment is to show that modeling the heat transfer of the heat sink as an *RC* circuit that is comprised of R_{ca} and C_{ca} is accurate enough. To perform this experiment we inserted external thermal sensor at the middle of the heat sink surface that measures, T_{ha} (temperature across the convective resistance, see Fig. 5). We run workload in a way that can capture the transient behavior of the package by starting at the idle state then at time 0 we execute two threads of *perl* (see Table II) for 600 s followed by another 600 s of idleness. To get representative measurements, the fans are kept at a default speed, which is about 25% of the max speed in our server. We set the local ambient to 24 °C to keep the machine cool enough so the fan speed stays fixed at the default value. Fig. 7 shows the measured and the modeled values of T_{ha} (referenced to its value when idle). The results clearly show a strong match between the real data and our *RC* circuit model. The transient behavior of T_{ca} is similar to T_{ha} except it has a higher amplitude due to the extra temperature across R_{hs} (see Fig. 5).

In the next experiment, we validate our assumptions and simulations that show the temperature of the heat sink is a function of the total power consumed in the CPU. We measure steady state temperature of T_{ha} (referenced to its

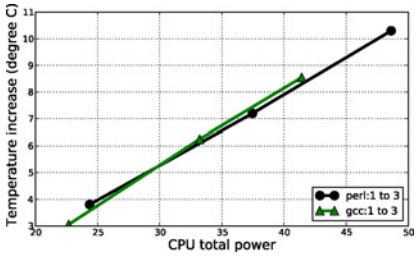

Fig. 8. T_{ha} versus CPU total power. T_{ha} is referenced to idle temperature.

value when idle) using the external thermal sensor as before. We execute one to three threads of two benchmarks, *perl* and *gcc*, where *perl* have higher core power than *gcc* (see Table II in Section VI). We use the same fan speed setup as in the previous experiment. The results in Fig. 8 clearly show that T_{ha} changes linearly with the total power despite us running two different workloads and varying the number of threads.

B. Sources of Cooling Savings at the Socket Level

Energy of cooling subsystems could be reduced by intelligently distributing the workload across the CPU sockets. To illustrate this, we use dual 4 core Intel Xeon sockets where each is associated with a fan. Two types of threads are executed, one highly active that consumes 14 W total power, and the other moderately active with 9.5 W. Temperature threshold is 85 °C, and local ambient temperature inside server is set to 42 °C. We use HotSpot for thermal simulation. Fig. 9 shows the impact of workload assignment on cooling cost savings at the socket level. The left part of the figure shows the thread assignments by state-of-the-art schedulers while the right part shows their assignments by our scheduling model. As shown in this figure, when there is a high imbalance in the total power between the sockets, we can intelligently balance power across the sockets and save on the cooling costs. The savings are 60%. We call this class of assignment *spreading*.

In the second scenario, the air-flow rate of socket 1 in the original assignment is about twice of that of socket 2. To minimize the cooling costs, we can swap the hot thread from socket 2 with two moderate threads from socket 1. Moving the hot thread to socket 1 does not increase the peak temperature since its power is similar to the power of hottest thread that is already there. In fact, the new assignment lowers the maximum temperature in socket 1 due to the reduction in total power by 5 W. The savings are 68%. We denote this class of assignments as *consolidation*.

However, a number of challenges need to be addressed to adopt a combination of spreading and consolidation. Workload migration across sockets comes with performance overhead of about 50–100 μs as we show later in Section VI-B3. We need to show that the time scale for cooling aware scheduling is orders of magnitude higher than the migration latency to ensure negligible performance overhead. The other challenge is to ensure a stable solution since the fan is a mechanical device and large variations in the fan speed impact its lifetime. Stable solution also helps to minimize the number of migration events between sockets. To solve this problem we use a control theoretic approach since ad-hoc solutions do not provide stability guarantees.

Fig. 9. Cooling aware scheduling at the socket level.

C. State-Space Controller and Scheduler

The first stage of our socket level management is the controller. It determines the amount of power that needs to be added/removed from the sockets to reduce cooling energy by balancing the fan speed when there is a large difference in the total power dissipation between CPU sockets. Such imbalance in fan speed is the source of energy inefficiency due to the cubic relation between fan power and its speed. The extra cooling cost consumed by a given socket, C_{cost_i}, can be computed as follows:

$$C_{cost_i} \sim (F_i^3 - F_{\text{avg}}^3) \tag{11}$$

where F_i and F_{avg} are the fan speed of the given CPU socket and target speed, respectively.

To design this controller we use state-space control since it is robust and scalable. We first extract the sate-space thermal model for the case to ambient temperature. The instantaneous value of case to ambient temperature for a given CPU, i, $T_{ca_i}(t)$, can be written as follows:

$$\frac{dT_{ca_i}(t)}{dt} = -\frac{T_{ca_i}(t)}{\tau_{ca_i}} + \frac{P_{\text{cpu}_i}(t)}{C_{ca_i}} \tag{12}$$

where τ_{ca_i} is the heat sink time constant, $\tau_{ca_i} = R_{ca_i} * C_{ca_i}$. The $P_{\text{cpu}_i}(t)$ represents the instantaneous power dissipated in the CPU socket. For the case of n number of CPUs, the vector of case to ambient temperatures can be written as $T_{ca} = [T_{ca_1}(t), T_{ca_2}(t), \ldots, T_{ca_n}(t)]^T$ and the vector of CPUs power can be expressed as $P_{\text{cpu}} = [P_{\text{cpu}_1}(t), P_{\text{cpu}_2}(t), \ldots, P_{\text{cpu}_n}(t)]^T$. For the case of n CPUs, (12) can be written as

$$\frac{dT_{ca}(t)}{dt} = YT_{ca}(t) + ZP_{\text{cpu}}(t) \tag{13}$$

where Y and Z are diagonal matrices that can be written as follows:

$$Y = \begin{pmatrix} -\frac{1}{\tau_{ca_1}} & & 0 \\ & \ddots & \\ 0 & & -\frac{1}{\tau_{ca_n}} \end{pmatrix} \qquad Z = \begin{pmatrix} \frac{1}{C_{ca_1}} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{C_{ca_n}} \end{pmatrix}.$$

The continuous system given in (13) can be discretized using the transformations given in [29] as follows:

$$T_{ca}(k+1) = \Phi T_{ca}(k) + \Gamma P_{\text{cpu}}(k) \tag{14}$$

where $\Phi = e^{Y\Delta t}$ and $\Gamma = Z\int_0^{\Delta t} e^{Yu}\,du$. The value of Δt is the sampling time. Since Y and Z are both diagonal matrices, Φ and Γ can be computed fairly easily as follows:

$$\Phi = \begin{pmatrix} e^{-\frac{\Delta t}{\tau_{ca_1}}} & & 0 \\ & \ddots & \\ 0 & & e^{-\frac{\Delta t}{\tau_{ca_n}}} \end{pmatrix}$$

$$\Gamma = \begin{pmatrix} \frac{1}{C_{ca_1}}\left(\int_0^{\Delta t} e^{-\frac{u}{\tau_{ca_1}}}\,du\right) & & 0 \\ & \ddots & \\ 0 & & \frac{1}{C_{ca_n}}\left(\int_0^{\Delta t} e^{-\frac{u}{\tau_{ca_n}}}\,du\right) \end{pmatrix}$$

where the integrals in Γ have a simple analytical solutions as $\frac{1}{C_{ca_i}}(\int_0^{\Delta t} e^{-\frac{u}{\tau_{ca_i}}}\, du) = R_{ca_i}(1 - e^{-\frac{\Delta t}{\tau_{ca_i}}})$.

To design the controller, we use standard control law that is based on the feedback of the system states [29]. Applying the control law to the state-space model given in (14) yields

$$P_{\text{cpu}}(k) = -GT_{ca}(k) + G_o T_{\text{ref}}(k) \tag{15}$$

where G and G_o are diagonal matrices of state feedback gain and input gain, respectively. T_{ref} corresponds to the vector of target temperatures. Using (14) and (15), the closed-loop heat sink temperature can be written as follows:

$$T_{ca}(k+1) = (\Phi - \Gamma G)T_{ca}(k) + \Gamma G_o T_{\text{ref}}(k). \tag{16}$$

This system is stable when the eigenvalues of $(\Phi - \Gamma G)$ are within a unit circle. We can find G by setting the eigenvalues to the desired values where the criteria for their selection are discussed shortly in this section. To compute the elements of G_o, we use steady-state analysis that requires the value of $G_{o_{ii}}$ to be equal to $(R_{ca_i}^{-1} + G_{ii})$, so the closed-loop system can settle at T_{ref_i}.

The T_{ref_i} can be computed based on the difference between the current fan speed and the target speed as described earlier. We need to translate the speed difference into change in case to ambient temperature, $\Delta T_{ca_i}(k)$, so as $T_{\text{ref}_i}(k) = T_{ca_i}(k) + \Delta T_{ca_i}(k)$. The value of T_{ca_i} can be estimated through thermal sensors attached to the heat sink. To estimate $\Delta T_{ca_i}(k)$, we calculate the change in the CPU case to ambient resistance, $\Delta R_{ca_i}(k)$ that corresponds to the difference between the current fan speed and the target speed. The $\Delta T_{ca_i}(k)$ can be estimated easily based on Ohm's as $\Delta T_{ca_i}(k) = \Delta R_{ca_i}(k)\frac{T_{ca_i}(k)}{R_{ca_i}(k)}$. The final value of T_{ref_i} can be calculated as follows:

$$T_{\text{ref}_i}(k) = T_{ca_i}(k) + \Delta R_{ca_i}(k)\frac{T_{ca_i}(k)}{R_{ca_i}(k)}. \tag{17}$$

We compute the transient response time of the controller by converting (16) to z-domain as $zT_{ca_i}(z) = \frac{z\gamma_i T_{\text{ref}_i}(z)}{z-(1-\gamma_i)}$, where $\gamma_i = 1 - \Phi_{ii} - \Gamma_{ii}G_{ii}$. The value of zT_{ca_i} is equivalent to $T_{ca_i}(k+1)$. Based on this equation, we obtain the time constant of the state-space controller for each CPU as follows:

$$\tau_{ssc_i} = -\frac{-\Delta t}{ln(1-\gamma_i)} = -\frac{-\Delta t}{ln(\lambda_i)}. \tag{18}$$

This equation shows that the transient time is a function of the sampling time and the controller eigenvalue, λ_i. Using this equation, we can set Δt to a few seconds while keeping τ_{ssc_i} in the range of seconds that is sufficient in our case since the temperature of the heat sink changes very slowly. It is not a good idea to make τ_{ssc_i} too short since it can cause undesirable overshoot in the system.

D. CPU Socket Scheduling

The details of the socket scheduling are given in Algorithm 2. The ΔP_{cpu} in this algorithm refers to the vector of the differences between the requested power by the controller and current power of the CPUs while P_{thr_i} corresponds to the power of the ith thread in a given CPU. Individual scheduling events are allowed only when the predicted cooling savings are higher than a given threshold, $S_{\min}$. The concepts of this algorithm are discussed below.

Algorithm 2 Socket Level Scheduling

```
1:  Calculate ΔP_cpu and the set of P_thr for each CPU. Set Q as an empty queue
2:  *** Spreading ***
3:  for i in set of unmarked CPUs do
4:    for  j in unmarked threads in CPU i do
5:      dest ⇐ CPU index with max(ΔP_cpu)
6:      if (P_thr_j < −ΔP_cpu_i AND P_thr_j < ΔP_cpu_dest) then
7:        if CPU dest has idle core AND no core level migrations in CPU dest then
8:          Calculate cooling savings of migrating thread j to CPU dest
9:        else
10:         Calculate cooling savings of swapping thread j with the coolest unmarked
            thread from CPU dest
11:       end if
12:       if cooling savings > S_min then
13:         Enqueue this migration event Q and mark migrated threads
14:         Update ΔP_cpu and threads assignment
15:         Mark CPU i and dest
16:       end if
17:     end if
18:   end for
19: end for
20:
21: *** Consolidation ***
22: while (unmarked CPUs > 1) do
23:   H ⇐ index of unmarked CPU with max fan speed
24:   if (fans have different speed) then
25:     L ⇐ index of unmarked CPU with min fan speed
26:   else
27:     L ⇐ index of any unmarked CPU that is different from H
28:   end if
29:   Find the hottest thread in CPU L (h^L)
30:   Starting from the coolest thread in H, find the smallest set of threads (S_cool)
      with total power ≥ P_thr_hL (P_thr of each thread in H is < P_thr_hL), then mark
      CPU H
31:   if (S_cool is not empty) then
32:     Calculate cooling savings of swapping the threads in S_cool with h^L
33:     if cooling savings > S_min then
34:       Enqueue this migration event in Q, update threads assignment and then
          mark CPU L
35:     end if
36:   end if
37: end while
38: Execute all migration events in Q
```

1) *Spreading Step (Steps 3–19):* Intuitively, since the fan power increases exponentially with cooling capacity it is important to balance the heat generation between the sockets in order to lower their fan speed and obtain cooling savings. Spreading is effective as long as there is sufficient imbalance in power consumption between CPU sockets such that the additional spreading migrations can lead to a better balance in socket power and fan speed. When this difference become sufficiently small, then the fans of the respective CPUs have reached their minimum speed from the spreading point of view. The controller decides on the amount of power that needs to be added to/removed from the sockets at each scheduling tick. We spread the workload starting with cooler threads to have finer grain control of the total power on each socket. Before each migration we evaluate the cooling savings as described in Section V-E to prevent ineffective scheduling decisions.

2) *Consolidation Step (Steps 21–37):* On the contrary, consolidation focuses on the cases when the total power consumption between sockets is comparable. Intuitively, since there are only relatively few fan speed settings, it is possible that at a particular speed the fan is actually capable of cooling a bit higher on die power density than is currently present on the socket. In a situation where there is a slight power density imbalance between the two sockets, it is possible that by switching cool and hot threads the increase in power density

on one socket is not too high, thus keeping the fan speed constant, but the decrease on the other socket is just big enough enabling it to lower its fan speed, and as a result saving energy. Thus, our strategy is to identify a set of cool threads on one socket whose power density is similar to a hot thread running on another socket, and then to swap them if our estimates (see Section V-E) show that we will be able to save cooling energy.

Process variation can impact the thermal distribution in the CPU. Our algorithm is able to adapt to this as it takes its input from thermal sensors that sense the effect of any variation.

E. Estimating Cooling Savings

We can estimate cooling savings for a particular scheduling decision by predicting the resultant fans speeds. Let us assume that we need to migrate a thread that consumes power, P_{thr}, from CPU i to CPU j. To approximate the resultant fan speed of the source CPU we apply Ohms's law to estimate the new R_{ca_i} which can be translated into fan speed. Let us assume that the increase in R_{ca_i} due to the migration equals ΔR_{ca_i}. The value of ΔR_{ca_i} can be estimated assuming the migrated thread is not the hottest in CPU i as follows:

$$\Delta R_{ca_i} = \frac{P_{\text{thr}} R_{sa_i}}{P_{\text{cpu}_i} - P_{\text{thr}}} \tag{19}$$

where $R_{sa_i} = R_{ca_i} + R_{s_i}$. If the migrated thread is the hottest in CPU i, then $\Delta R_{ca_i} = \frac{P_{\text{thr}} R_{sa_i} + R_{\text{core}_i} \delta p_{\text{thr}_i}}{P_{\text{cpu}_i} - P_{\text{thr}}}$, where δp_{thr_i} represents the core power difference between the hottest thread that we migrate and the second hottest thread in the CPU i. For the socket that is receiving the extra thread, there are few cases that need to be considered.

1) *Case A:* The first case is when the migrated thread dissipates less core power than the hottest thread in the destination socket and the destination's fan is not idle. Using Ohm's law, the value of its ΔR_{ca_j} can be computed simply as follows:

$$\Delta R_{ca_j} = -\frac{P_{\text{thr}} R_{sa_j}}{P_{\text{cpu}_j} + P_{\text{thr}}}. \tag{20}$$

2) *Case B:* This case addresses the scenarios when the newly migrated thread dissipates more core power than the hottest thread in the destination socket and the destination's fan is not idle. When adding a thread that consumes more power than the current hottest thread, we expect maximum temperature increases not only in the heat sink and heat spread but also in the core. The value of ΔR_{ca_j} is as follows:

$$\Delta R_{ca_j} = -\frac{P_{\text{thr}} R_{sa_j} + R_{\text{core}}(P_{\text{thr}}^{\text{core}} - P_{\text{thr}_j}^{\max})}{P_{\text{cpu}_j} + P_{\text{thr}}} \tag{21}$$

where $P_{\text{thr}_j}^{\max}$ corresponds to the core power of the hottest thread in the destination CPU j. $P_{\text{thr}}^{\text{core}}$ refers to the core power component of the migrated thread.

3) *Case C:* The next case we need to consider is when the destination fan is idle. If the thread to be migrated has lower core power than the hottest thread $T_{\max_j}$, then adding the extra thread will increase the temperature of already existing hottest thread by $P_{\text{thr}} R_{sa_j}$. If the predicted core temperature after migration, $T_{\max_j}^{\text{new}} = T_{\max_j} + P_{\text{thr}} R_{sa_j}$, is found to exceed the fan trigger temperature, T_c^{fan}, then we can estimate ΔR_{ca_j} as follows:

$$\Delta R_{ca_j} = -\frac{T_{\max_j}^{\text{new}} - T_c^{\text{fan}}}{P_{migr} + P_{\text{cpu}_j}}. \tag{22}$$

4) *Case D:* The last case is when the power of the thread to be migrated exceeds the power of the hottest thread in the destination socket that has its fans idle. In this case, the maximum temperature in the destination CPU would be higher due to the increase in T_{ca_j} as well as the core and heat spreader temperatures. We can calculate the resultant ΔR_{ca_j} based on Ohm's law as before. We next present how we estimate the induced power by the active threads at low cost.

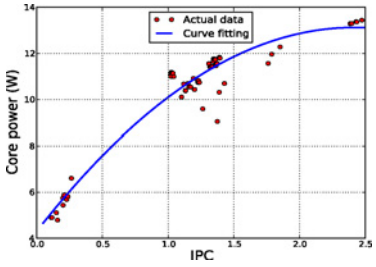

Fig. 10. Dynamic power model.

F. CPU Power Estimation

The CPU power can be divided into two basic components: dynamic power and leakage power [4]. The dynamic power is a function of the induced activity by the application on the components. On the other hand, the leakage power is function of components area and the current temperature. In the following, we describe how we estimate these values at small overhead. We need power traces for each core in our system in order to evaluate temperature and cooling savings.

To model the dynamic power for the CPU cores, we need a metric that is directly correlated with the level of activity in the core. For this, we use the number of retired instructions per cycle or IPC that is shown to have a good correlation with the CPU power [30]. Executing instructions cover almost all of the dynamic activities in the core. Additionally, the rate of the instruction execution is expected to have a sufficient correlation with the induced activity since the faster the execution the more activity there is.

Before we generate the model, we collect real-time power and IPC measurements data on state-of-the-art quad core dual socket Xeon E5440 server. The IPC traces are collected using processor performance counters (model specific registers). We generate the power traces as follows. We run the benchmarks in one of the cores and keep the rest in idle mode to get an accurate power trace per benchmark. We measure the power of the CPU package by inserting current sensors in the power connector of the CPU socket and collect power traces using data acquisition system. This power is composed of three main components: CPU baseline power, core power, and L2 cache power. To exclude the CPU baseline power we measure the CPU power when it is idle and then subtract this baseline power from the CPU socket power trace. We deactivate the CPU C-states during the baseline power measurements as otherwise the CPU socket would go to a deep C-state when it is

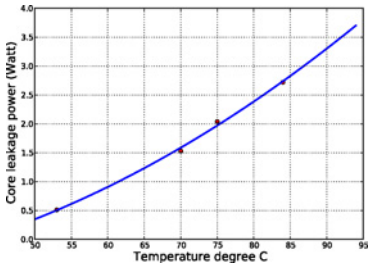

Fig. 11. Leakage power model.

not executing. The power of the idle cores is already accounted for in the baseline power measurement. The resultant trace represents the core and L2 cache power components, which we call *core+L2* trace. Subsequently, we estimate the L2 power based on its access rate since it is a regular structure. We convert the L2 access rate to power using Cacti tool that is widely used for estimating the power of the cache subsystem [31], [32]. The trace of L2 access rate is extracted during execution using CPU performance counters. We isolate the core power trace by subtracting the L2 power from core+L2 trace.

To generate the dynamic power model we ran five SPEC2000 benchmarks that span from memory intensive to CPU intensive applications. For each benchmark we collected ten power and IPC samples that are sufficient since the benchmarks behavior is reasonably stable. To generate the model we use regression to find the best fit for the samples using *polyfit* function in python programming language. Fig. 10 shows the dynamic power model for the cores where polynomial of order 2 is used. The results show that using IPC metric exhibit a good correlation with the dynamic power. The average error is less than 1 W that is sufficient for our thermal modeling needs.

The leakage model is a function of the components area and the current temperature. To generate the leakage model for Xeon E5440, we run experiments by blocking the air flow to increase the CPU temperature and collecting the power measurements concurrently. Fig. 11 shows how the core leakage power changes with temperature and the model that we generated based on these measurements. We developed the leakage model using the regression method where the model can be represented by a simple second-order polynomial. The results show a good agreement between the modeling and the experimental outcomes.

Building these power models requires one time cost in terms of offline analysis. Once they are developed, they can be reused across the deployed servers with similar CPUs, hence this modeling approach is cost efficient. The runtime overhead of our models is negligible since computing a second-order polynomial can be done in few CPU cycles. In the next section, we provide experimental results to evaluate our algorithms.

VI. Evaluation

A. Methodology

We evaluate our approach using 45 nm Intel Quad Core dual socket Xeon E5440 server. CPU sockets share 32 GB main memory. The system runs the latest Linux kernel (2.6.33.2) that manages all system resources. We extract the core power dissipation traces of different benchmarks using measurements on our machine. We run the benchmarks in one of the cores and keep the rest in idle mode to get an accurate core and L2 power traces per benchmark using the method that is described in Section V-F. The core and L2 power traces are used to estimate the core and L2 temperatures, respectively, using HotSpot simulator [26] and the Xeon processor layout. We also extract the baseline power of the CPU using the method in Section V-F. We include the baseline power of the CPU in the temperature simulations since its impacts the heat spreader and heat sink temperature. We also account for leakage power that we compute using our model that we discussed in Fig. 11.

We use simulation instead of running our algorithms in real system due to a number of issues. The built-in fan control algorithm runs all the CPU fans at a single speed that is related to cooling needs of the hottest socket, which lead to over provisioning. The algorithm is implemented in a separate controller that we do not have access to. Consequently, not all the benefits of our algorithms can be manifested with using the built-in fan control algorithm. HotSpot is also necessary to conduct experiments that require setting different fan speeds per socket because the hardware does not currently support such fine grained control. HotSpot enables us to profile and study the temperature breakdown in the CPU and thermal package as a function of fan speed.

Quad Core Xeon E5440 package characteristics that have been used in thermal simulation are listed in Table III. For the baseline *fan control algorithm*, we assumed a closed loop controller similar to that used in modern systems. The fan algorithm adjusts the fan speed in proportion to the heat level of their respective CPUs. When the temperature is below a given threshold the fan is set to idle speed. We set the fan to trigger 3° below the CPU threshold temperature to allow for enough time to react to thermal emergencies since it is a mechanical device. The fan speed in real systems normally change in steps that makes it prone to small fluctuations. We mitigate these fluctuations by the use of a guard window where the fans stay at a same speed within this window. We set the guard window to seven CFM. This fan control algorithm is used for all the policies that we examine.

The critical temperature is set to 85 °C. In cases when the temperature exceeds the CPU critical temperature, we use throttling mechanism as back up. We assume there is a support for accurate temperature sensors reading in the system. Recent work [33], [34] has proposed efficient techniques to estimate accurate temperature in presence of noisy sensors. We consider the local ambient temperature inside the server to be in a range between 36 °C and 42 °C that is reasonable assumption due to the following reasons. The servers inlet temperatures can be much higher than the room ambient and reach 40 °C [20]. The new trend in data centers is to increase the ambient temperature to lower the costs of the air conditioning system [21], which would result in even higher inlet temperatures. Since the local ambient within the server is hotter than the inlet temperature, it is not uncommon for the local ambient temperature to reach up to 45 °C [26], [28].

Core level thermal management polices perform scheduling at each OS tick. To guard against ineffective migrations, we

TABLE II
SPEC BENCHMARKS CHARACTERISTICS

Benchmark	Dynamic Core Power (W)
perl	12.7
bzip2	11.6
eon	11.2
twolf	11.1
gzip	10.4
gcc	10.2
mcf	5.47

TABLE III
CHARACTERISTICS OF QUAD CORE XEON E5440 PACKAGE

Heat spreader thickness	1.5 mm
Case to ambient thermal resistance, R_{ca} K/W	$R_{ca} = 0.141 + \frac{1.23}{V^{0.923}}$, V: air flow in CFM [22]
Max. air-flow rate per socket	53.4 CFM [35]
Fan power per socket	29.4 W [35]
Fan steps	32
Idle fan speed	10% of max speed

allowed migration between cores to be initiated only when the temperature difference between the hot core and colder core is above 5 °C. For socket level thermal management, we trigger scheduling every 4 s, since heat sink temperature changes slowly (measured average thermal time constant is about 50 s). We set the cooling savings threshold to a conservative value of 10%. In our results, we account for the temperature warm-up to get accurate results.

Benchmarks from the SPEC2000 suite have been used as workloads (see Table II). A set of benchmarks are selected that exhibit various levels of CPU intensity to emulate real-life applications. We run each benchmark in the workset till its completion and repeat it until the end of simulation time. The evaluation time for our algorithms is set to 40 s after the warm-up period. We also use synthetic benchmarks to test the transient behavior of our socket level scheduler.

We evaluate our core level and multitier algorithms. Our CPTM reduces the cooling costs by minimizing the hot spots between cores. The MTTM performs core level and socket level thermal management to reduce the hot spots within and between sockets. We compare these algorithms against the following set of state-of-the-art policies.

1) *DLB* is usually implemented in modern operating systems to enhance the utilization of the system resources. The DLB performs thread migration to lower the difference in task queue lengths of the individual cores [9]. The operating system initiates dynamic load balancing every hundred milliseconds. In this paper, we implement the DLB as our *default policy* for the purposes of comparison.
2) *Core level reactive thermal management (CRTM):* This policy is similar to our CPTM except that the thermal scheduling decisions are reactive, are based on the current temperature rather than predicted ones.
3) *Basic socket level thermal management (BSTM):* When there is imbalance in the fan speed, the policy migrates the hottest thread from the socket that is associated with the highest fan speed to the socket with the lowest fan speed. We set the scheduling period to be similar to that of our algorithm, MTTM, to eliminate fan instability that can result from poor selection of the scheduling period.

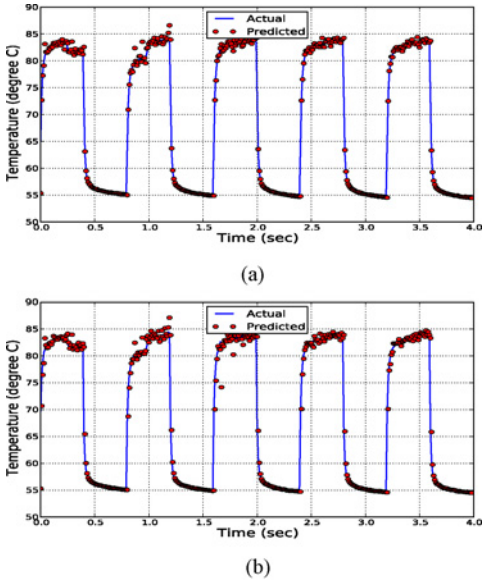

Fig. 12. Temperature prediction using BLP and running *gcc* benchmark. (a) Prediction distance of 5 ms. (b) Prediction distance of 10 ms.

B. Results

To quantify and understand clearly the benefits of our algorithms, first, we evaluate our core level algorithm. Then, we study our multilevel technique. At the end, we discuss the overhead associated with our algorithms. The results show that our technique gives average cooling energy savings of 80%.

1) *Core Level Thermal Management:* We start with evaluating our BLP. In order for the predictor to be feasible, we need to show that the prediction window is in the range of operating system scheduling period (1–10 ms). The feasible parameters that allow for a prediction distance to be around the range of OS scheduling period are $m = 3$, $\alpha = (0.10–0.13)T_s$, $N = 3$, and sampling time (T_s) is a factor of τ depending on the prediction span. The value of α is chosen as a tradeoff between prediction distance and accuracy. This is because smaller values of α lead to larger coefficients that make the predictor more prone to noise. The selected value of $N = 3$ is based on numerous simulations that we have conducted. We found that larger values of N may compromise accuracy since the predictor become more susceptible to noise and the high-frequency harmonics in the signal that cannot be filtered completely.

In Fig. 12, we show an illustrative example of applying the BLP to predict the temperature from executing real benchmark. In this experiment, we chose *perl*, a CPU intensive workload form SPEC2000 suite. To make the temperature profile harder to predict, we run the application for a period of 400 ms followed by idle interval of 400 ms and repeat the sequence to induce large variations in the temperature profile. In this example, we assume $\alpha = 0.11T_s$. Fig. 12(a) shows the prediction results with 5 ms prediction span. It is clear from the figure that BLP is fairly accurate in predicting most parts of the signal with a small average of error of 0.52 °C despite the large variations in the temperature signal. Fig. 12(b) shows the prediction profile with prediction distance of 10 ms, a value that represent the high side of the operating system scheduling period. The results in this figure show that our predictor is able to sustain a decent prediction accuracy in this range

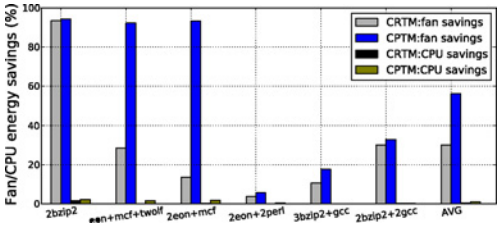

Fig. 13. Fan and CPU energy savings using core level polices.

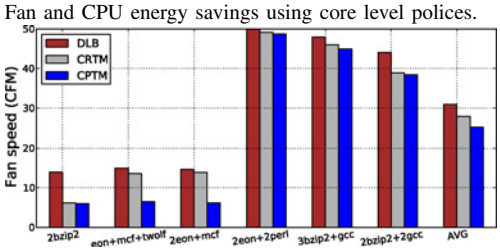

Fig. 14. Fan speed reduction with core level thermal management polices.

with average error equal to 0.85 °C only. In the rest of our experiments, we use a conservative prediction distance of 9 ms.

To quantify the benefit of our core level approach, we run various combination of workloads in a single socket. We set the local ambient temperature to 42 °C. The results in Fig. 13 show the energy savings of CRTM and CPTM compared to the default policy, DLB. The results show that using CPTM provides significant cooling savings over the other techniques, the average enhancement over the base line policy of 56%. The savings result from reducing fan speed across all combinations. As it can be seen from the results, the cooling savings are higher at lower utilization or with more heterogeneity in the workload. Execution at lower utilization gives more room for distributing the heat between cores, thus reducing thermal emergencies. For the case of 50% utilization, $2bzip2$, the cooling savings reaches 95%. The other case of $2eon + mcf$ is an example of heterogeneous workload that is composed of cpu intensive, *eon*, and memory intensive *mcf* jobs and has a higher utilization factor of 75%. The reported results show that CPTM is able to deliver 93.3% cooling savings. CPTM is able to provide nice cooling savings with heterogeneous workload even when the processor is executing four threads as shown in the case of $2bzip2 + 2gcc$. The limitation of CPTM is shown in the case when the socket is running four threads that are all CPU intensive, $2perl + 2eon$. The cooling savings in this case is small since there is a little room for better thermal balancing as all cores are heavily used. We will show shortly that such cases can be mitigated by using our socket level scheduling approach because it expands the scope of thermal management and provides more opportunities for better thermal distribution. On the other hand, the CPU socket energy savings due to lowering the leakage power is low; the average improvement using CPTM over the baseline policy is about 1.0% which is slightly higher than the case of CRTM. The reason for the low savings is related to the fan contribution of keeping the temperature around the target threshold for both baseline and core level policies, which minimizes the difference in leakage power between these polices.

Fig. 14 shows the cooling rates using all three policies: 1) DLB; 2) CRTM; and 3) CPTM. As the results clearly show, applying CPTM outperforms all other techniques and gives

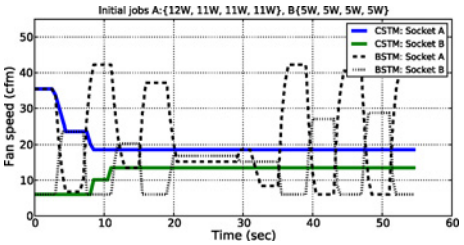

Fig. 15. Workload spreading.

appreciable reduction in required cooling rate; the average improvement over the based line policy is 19%. Similar to the case of energy savings, the cooling rate reduction is higher at lower utilization or with high heterogeneity in the workload. For the case of 50% utilization, $2bzip2$, is able to keep the fan at idle speed while using DLB requires the fan to run at more than twice the idle speed. For the cases with heterogeneous workloads (e.g., $2eon + mcf$), the results show that CPTM is able to eliminate almost all thermal emergencies and keep the fan very close to idle speed while both DLB and CRTM require the fan to run at much higher speeds. The savings in the cases of running four threads when all benchmarks are CPU intensive is small since all cores are used intensively. Such cases can be mitigated with our socket level approach as we show below.

2) *Multitier Thermal Management:* We start with addressing the stability of our socket scheduling then we study the cooling savings of our multitier thermal management algorithm.

In our stability study, we compare our control-theoretic socket level thermal management (CSTM) against the BSTM. We study two situations, one where spreading strategy gives a good solution and the other where consolidation is more applicable. Fig. 15 shows the spreading case with the following initial job assignment, socket A: {12 W + 11 W + 11 W + 11 W} and socket B: {5 W + 5 W + 5 W + 5 W} and local ambient temperature of 41 °C. The efficient solution for this case is to have the two sockets run a better balanced workload as A: {12 W + 5 W + 11 W + 5 W} and B: {11 W + 5 W + 11 W + 5 W}. The CSTM controller converges to the target solution in two scheduling ticks only. In contrast, BSTM policy fails to converge. The transient behavior of a consolidation case is shown in Fig. 16. The initial workload assignment for sockets A and B is {13 W + 8 W + 8 W} and {13 W + 7 W + 7 W}, respectively. Our controller swaps the 13 W thread from socket B with two threads of 8 W from socket A and converges to the solution in one scheduling tick. In this case, the fan of socket A slightly reduces while the fan of socket B drops to idle speed. On the other hand, the BSTM fails to reach a stable solution. The results indicate that our controller is able to converge to the target solution while BSTM heuristic solution fails in this aspect.

To evaluate our MTTM we use a combination of workloads with various thermal stress levels and utilization values. The list of the workload combinations that we use is given in Table IV. Fig. 17 shows the cooling energy savings of CRTM, CPTM, and MTTM compared to the default DLB policy. The results show that using MTTM provides substantial savings over the other techniques. The average improvement over the default policy is 80%. Savings come from spreading

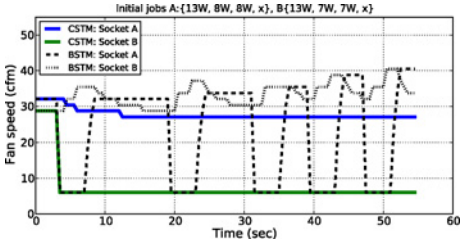

Fig. 16. Workload consolidation.

TABLE IV
WORKLOAD COMBINATIONS FOR MULTITIER ALGORITHM

Workload	Socket A	Socket B	Local Ambient (°C)
W1	*3eon*	*eon + mcf + gcc*	42
W2	*2eon + mcf*	*eon + bzip2 + mcf*	42
W3	*2bzip2 + 2mcf*	*2bzip2 + 2mcf*	42
W4	*2perl + 2eon*	*2gcc + 2mcf*	42
W5	*2perl + 2bzip2*	*2gcc + 2mcf*	39
W6	*2perl + 2bzip2*	*2gcc + 2mcf*	36
W7	*2perl + bzip2*	*gcc+2mcf*	42
W8	*perl + 3gcc*	*perl + 2gcc*	42
W9	*perl + 3gcc*	*perl + 2gcc*	40
W10	*perl + 3gcc*	*perl + 2gcc*	38
W11	*perl + 3gzip*	*perl + gcc + gzip*	42
W12	*3eon*	*3eon*	42
W13	*2eon+mcf*	*2eon+mcf*	42
W14	*2bzip2*	*2bzip2*	42

and consolidation at the socket level and proactive thermal management at the core level. The savings in the example of workload W5 are due to workload spreading. In this case, socket A has high thermal stress while socket B remains under low to medium stress. Under such scenario core level management becomes ineffective since all threads in socket A have high power density, so there is not enough room to spread the heat. However, workload spreading at the socket level is highly effective because it can balance the temperature between the two sockets. The savings in this case are 58.3%. Workloads W8 and W11 get savings primarily from consolidation. More specifically, in case of W8 the scheduler swaps *perl* with two instances of *gcc*. CPTM plays a big role when socket level scheduling becomes ineffective. This is apparent in the case of workload W13 where the savings come from CPTM since the workload between the two sockets is initially balanced.

We also study the impact of changing local ambient temperature inside the server on relative savings over the default policy. In the first case, we reduce the local ambient temperature from 39 °C (W5) to 36 °C (W6) where the actual workload stays the same. The results show that MTTM savings over the default policy is improved at 36 °C due to the extra savings from the CPTM policy. Similar behavior can be seen in the cases $\{W8, W9,$ and $W10\}$ where the ambient changes from 42 °C to 38 °C. Our multitier thermal management effectively mitigates thermal problems when executing workloads with highly diverse thermal characteristics.

Fig. 18 shows the maximum performance loss per core due to thermal emergencies when the fan cooling is insufficient or when the fan response is slow. The results show that employing MTTM keeps the percentage of thermal emergencies below 1%, while for the other policies DLB, CRTM, and CPTM can reach 4.5%, 2.5%, and 2.5%, respectively. This advantage of MTTM come from managing the temperature at multiple levels.

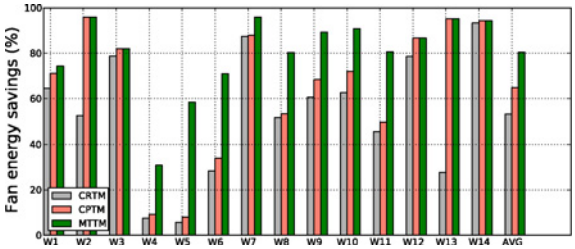

Fig. 17. Energy savings using multitier thermal management.

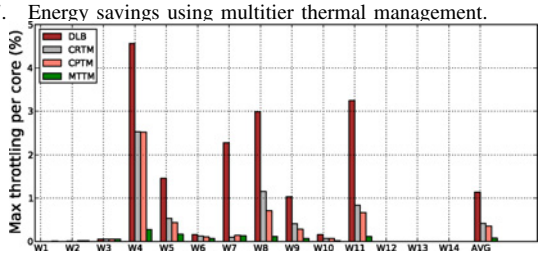

Fig. 18. Reducing thermal emergencies.

3) *Overhead:* The performance overhead of our multitier thermal management can broken down into the overhead of the core and the socket levels. The migration overhead between cores is in the range of few microseconds, e.g., 5 μs [23], which is much smaller compared to the time between OS scheduling ticks. The primary cause of this overhead is the warm up of the private cache of the cores. The computational overhead is negligible because new temperature prediction takes only few CPU cycles. The socket level migration overhead includes both thread migration and scheduler computational costs. When a thread migrates from a socket to another it needs to warm up the cache subsystem of the destination CPU socket. The memory subsystem is shared among the CPU sockets in our server that implements a single coherent memory system, where memory coherency is enforced by the hardware. Warming up the cache subsystem is the primary source of the migration overhead. This warm up time includes updating both L2 cache of the CPU and the private cache of the destination core as well. The overall warm up time increases by 10–20X compared to core level migration warm up. This is because the L2 needs to bring the data of the newly migrated thread from the memory this costs around 100–200 cycles (e.g., 150 cycles [23]). The overall migration time is on the order of 50–100 μs that is included in our results. Despite this extra overhead, it is still orders of magnitude smaller than the period between the two scheduling ticks, which is on the order of seconds. The socket scheduler executes ordinary computations that require no more than several milliseconds. This analysis shows that our multitier thermal management is a light weight technique that makes it applicable to real systems.

VII. CONCLUSION

With the ever increasing computational demand, the high-end servers undergo tremendous thermal stress that has detrimental effect on the system characteristics. Many of the proposed DTM techniques do not model system cooling dynamics and were limited to single socket CPU systems. Consequently, these techniques compromised system performance and provided substandard system power optimization.

In this paper, we proposed a new multitier workload scheduling technique that minimizes the cooling energy costs

of fan subsystem in a multisocket CPU platform. At the core level, a new band limited predictor was employed for proactive thermal management that minimizes the cooling costs and reduces the frequency of thermal emergencies. Our predictor is not only accurate but has very low computational cost and it does not require runtime adaptation. The top-level scheduler leverages freedom in workload assignment at socket level and helps achieve better thermal distribution via spreading and consolidation. It employed our provably stable control theoretic framework for scheduling the workload with negligible performance overhead. The reported results showed that our approach delivers 80% average cooling energy savings compared to the state-of-the-art policies.

REFERENCES

[1] *Sun Fire X4270 Server* [Online]. Available: www.sun.com/servers/x64/x4270

[2] *Intel® Server Board S5400SF* [Online]. Available: www.intel.com/products/server/motherboards/s5400sf

[3] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *Proc. Int. Symp. Comput. Architect.*, 2003, pp. 2–13.

[4] M. Pedram and S. Nazarian, "Thermal modeling, analysis, and management in VLSI circuits: Principles and methods," *Proc. IEEE*, vol. 94, no. 8, pp. 1487–1501, Aug. 2006.

[5] A. Ajami, K. Banerjee, and M. Pedram, "Modeling and analysis of nonuniform substrate temperature effects on global interconnects," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 6, pp. 849–861, Jun. 2005.

[6] M. Patterson, "The effect of data center temperature on energy efficiency," in *Proc. Intersoc. Conf. Thermal Thermomech. Phenomena Electron. Syst.*, 2008, pp. 1167–1174.

[7] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. Keller, "Energy management for commercial servers," *IEEE Comput.*, vol. 39, no. 12, pp. 39–48, Dec. 2003.

[8] R. Lyon and A. Bergles, "Noise and cooling in electronics packages," *IEEE Trans. Components Packag. Technol.*, vol. 29, no. 3, pp. 535–542, Sep. 2006.

[9] J. Choi, C. Cher, H. Franke, H. H. A. Weger, and P. Bose, "Thermal-aware task scheduling at the system software level," in *Proc. Int. Symp. Low Power Electron. Des.*, 2007, pp. 213–218.

[10] A. Coskun, T. Rosing, and K. Gross, "Proactive temperature management in MPSoCs," in *Proc. Int. Symp. Low Power Electron. Design*, 2008, pp. 165–170.

[11] A. Kumar, L. Shang, L. Peh, and N. Jha, "Hybdtm: A coordinated hardware–software approach for dynamic thermal management," in *Proc. Des. Automat. Conf.*, 2006, pp. 548–553.

[12] I. Yeo, C. Liu, and E. Kim, "Predictive dynamic thermal management for multicore systems," in *Proc. Des. Automat. Conf.*, 2008, pp. 734–739.

[13] D. Mugler, "Computationally efficient linear prediction from past samples of a band-limited signal and its derivative," *IEEE Trans. Informat. Theory*, vol. 36, no. 3, pp. 589–596, May 1990.

[14] D. Mugler, "Computational aspects of an optimal linear prediction formula for band-limited signals," *Comput. Appl. Math.*, pp. 351–356, 1992.

[15] F. Marvasti, *Nonuniform Sampling Theory and Practice*. New York: Kluwer/Plenum, 2001.

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

[17] Z. Wang, C. Bash, N. Tolia, M. Marwah, X. Zhu, and P. Ranganathan, "Optimal fan speed control for thermal management of servers," in *Proc. IPAC*, 2009, pp. 1–10.

[18] H. Chiueh, L. Luh, J. Draper, and J. Choma, "A novel fully integrated fan controller for advanced computer systems," in *Proc. Southwest Symp. Mixed-Signal Des.*, 2000, pp. 191–194.

[19] D. Shin, J. Kim, J. Choi, S. Chung, and E. Chung, "Energy-optimal dynamic thermal management for green computing," in *Proc. Int. Conf. Comput.-Aided Des.*, 2009, pp. 652–657.

[20] R. Schmidt, E. Cruz, and K. Iyengar, "Challenges of data center thermal management," *IBM J. Res. Develop.*, vol. 49, nos. 4–5, pp. 709–723, 2005.

[21] *Reducing Data Center Cost with an Air Economizer*, Intel, 2007.

[22] *Quad-Core Intel Xeon Processor 5300 Series: Thermal/Mechanical Design Guidelines*, Intel, 2007.

[23] G. Mohamed, M. Powell, and T. Vijaykumar, "Heat-and-run: Leveraging SMT and CMP to manage power density through the operating system," *SIGOPS Oper. Syst. Rev.*, vol. 38, no. 5, pp. 260–270, 2004.

[24] D. Mugler and Y. Wu, "An integrator for time-dependent systems with oscillatory behavior," *Comput. Methods Appl. Mechanic. Eng.*, vol. 171, nos. 1–2, pp. 25–41, 1999.

[25] A. Papoulis, *Signal Analysis*. London, U.K.: McGraw-Hill, 1981.

[26] K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," in *Proc. TACO*, 2004, pp. 94–125.

[27] P. Chaparro, J. Gonzalez, G. Magklis, Q. Cai, and A. Gonzalez, "Understanding the thermal implications of multicore architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 8, pp. 1055–1065, Aug. 2007.

[28] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware computer systems: Opportunities and challenges," *IEEE Micro.*, vol. 23, no. 6, pp. 52–61, Nov.–Dec. 2003.

[29] G. Franklin, J. Powel, and M. Workman, *Digital Control of Dynamic Systems*. Reading, MA: Addison-Wesley, 1990.

[30] G. Dhiman, K. Mihic, and T. Rosing, "A system for online power prediction in virtualized environments using Gaussian mixture models," in *Proc. Des. Automat. Conf.*, 2010, pp. 807–812.

[31] D. Tarjan, S. Thoziyoor, and N. Jouppi, "Cacti 4.0," HP Laboratories, Palo Alto, CA, Tech. Rep. HPL-2006-86, 2006, pp. 1–15.

[32] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies," in *Proc. Int. Symp. Comput. Architect.*, 2008, pp. 51–62.

[33] 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.

[34] Y. Zhang and A. Srivastava, "Accurate temperature estimation using noisy thermal sensors," in *Proc. Des. Automat. Conf.*, 2009, pp. 472–477.

[35] *Sunon 40x40x56 mm, 26.7 CFM* [Online]. Available: http://www.sunon.com.tw/products/pdf/DCFAN/PMD4056.pdf

Raid Ayoub (S'09) received the B.S. and M.S. degrees in electrical engineering from the University of Technology, Baghdad, Iraq. He is currently pursuing the Ph.D. degree in computer engineering from the Department of Computer Science and Engineering, University of California at San Diego, La Jolla.

His current research interests include system level design and computer architecture with a specific focus on energy, thermal, and cooling management.

Krishnam Indukuri (M'11) received the B.E. (honors) degree in electrical and electronics engineering from Birla Institute of Technology and Science-Pilani, Pilani, India, in 2007, and the M.S. degree in electrical and computer engineering from the University of California at San Diego, San Diego, in 2010.

He is currently with Qualcomm, Inc., San Diego. His current research interests include low-power architectures for system design.

Tajana Simunic Rosing (M'01) received the M.S. degree in electrical engineering from the University of Arizona, Tucson, on high-speed interconnect and driver–receiver circuit design, and the Ph.D. degree from Stanford University, Palo Alto, CA, in 2001, concurrently with finishing the Masters in engineering management on dynamic management of power consumption.

Prior to pursuing the Ph.D., she was a Senior Design Engineer with Altera Corporation, San Jose, CA. She was a Full-Time Researcher with HP Laboratories, Palo Alto, CA, while working part-time at Stanford University. At Stanford University, she led the research of a number of graduate students and has taught graduate-level classes. She is currently an Assistant Professor with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla. Her current research interests include energy-efficient computing, embedded, and wireless systems.

Dr. Rosing has served on a number of technical paper committees. She is currently an Associate Editor of the IEEE TRANSACTIONS ON MOBILE COMPUTING.