Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs

Ayşe Kıvılcım Coşkun, *Student Member, IEEE*, Tajana Šimunić Rosing, *Member, IEEE*, and Kenny C. Gross, *Member, IEEE*

***Abstract*—Conventional thermal management techniques are reactive, as they take action after temperature reaches a threshold. Such approaches do not always minimize and balance the temperature, and they control temperature at a noticeable performance cost. This paper investigates how to use predictors for forecasting temperature and workload dynamics, and proposes proactive thermal management techniques for multiprocessor system-on-chips. The predictors we study include autoregressive moving average modeling and lookup tables. We evaluate several reactive and predictive techniques on an UltraSPARC T1 processor and an architecture-level simulator. Proactive methods achieve significantly better thermal profiles and performance in comparison to reactive policies.**

***Index Terms*—Energy management, prediction methods, processor scheduling, temperature control, thermal factors.**

I. Introduction

CHIP POWER consumption is expected to increase with each new generation of computers while the geometries continue to shrink, resulting in higher power densities. High power densities cause thermal hot spots and large temperature variations on the die and introduce a number of significant challenges, including higher cooling costs, increase in leakage power, lower device performance, and reliability degradation [18], [30]. In this paper, we show that, by performing thermal management and job allocation on a multiprocessor system-on-chip (MPSoC) proactively, thermal emergencies and temperature-induced problems can be avoided. The proactive thermal management techniques we propose utilize predictors for forecasting the thermal or workload dynamics on cores. Based on the predictions, our techniques are then able to act on thermal hot spots and temperature variations ahead of time.

Previously proposed thermal management techniques are typically activated upon reaching a temperature threshold, and they maintain the temperature below the critical levels at a considerable performance cost [28]. In multiprocessor domain, techniques such as thread migration and proportional–integral–derivative control [11] or temperature-aware job scheduling [9] have been introduced to achieve a safe die temperature at a reduced performance impact. Still, such techniques are reactive in nature, i.e., they also take action after the temperature reaches a predetermined level. Furthermore, conventional dynamic thermal management techniques do not focus on balancing the temperature, and as a result, they can create large spatial variations in temperature or thermal cycles. In particular, in systems with dynamic power management (DPM) that turns off cores, reliability degradation can be accelerated because of the cycles created by workload rate changes and power management decisions [24].

In this paper, we extend the proactive thermal management technique we proposed in [8]. The goal of our technique is to prevent thermal problems at very low performance overhead. In our experiments, we have seen that, as the workload goes through stationary phases, temperature can be estimated accurately by regressing the previous measurements. Thus, we utilize autoregressive moving average (ARMA) modeling for estimating future temperature accurately based on temperature measurements. Since our goal is to proactively allocate workload, it is essential to detect the changes in workload and temperature dynamics as early as possible and adapt the ARMA model if necessary. For detecting these changes at runtime, we use sequential probability ratio test (SPRT), which provides the earliest possible detection of variations in time series signals [34]. The early detection of variations enables us to update the ARMA model for the current thermal dynamics immediately and avoid inaccuracy.

Utilizing the forecast, our *proactive temperature balancing* (*PTB*) technique allocates incoming jobs to cores to minimize and balance the temperature on the die. In multithreaded systems, PTB performs balancing by first moving threads that are currently waiting, as opposed to stalling the executing threads. In single-threaded environments, we bound the number of thread reallocations to reduce the performance cost. In comparison to the state-of-art allocation methods used in modern operating systems (OSs), our technique's performance overhead is negligible, while it effectively reduces the frequency of hot spots and gradients. For example, on an UltraSPARC T1 processor [21], PTB achieves 60% reduction in hot spot occurrences, 80% reduction in spatial gradients, and 75% reduction in thermal cycles on average in comparison to reactive thermal management, while incurring a performance cost of less than 2% with respect to the default scheduling policy running on the system. This paper extends our previous work in [8] in the following directions.

1) We provide an extensive study on how to use various predictors (ARMA, history predictor [17], exponential

Manuscript received November 29, 2008; revised March 2, 2009 and May 11, 2009. Current version published September 18, 2009. This work was supported in part by Sun Microsystems, by UC MICRO, by the Center for Networked Systems (CNS) at UCSD, by the MARCO/DARPA Gigascale Systems Research Center, and by NSF Greenlight. This paper was recommended by Associate Editor D. Atienza.

A. K. Coşkun and T. Š. Rosing are with the Department of Computer Science and Engineering, University of California San Diego, La Jolla, CA 92093-0404 USA (e-mail: acoskun@ucsd.edu; trosing@ucsd.edu).

K. C. Gross is with Sun Microsystems, San Diego, CA 92121-1973 USA (e-mail: kenny.gross@sun.com).

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.2009.2026357

averaging, and recursive least squares predictor [35]) to forecast temperature and workload dynamics. We compare these techniques' prediction accuracy, overhead, and adaptation capabilities.

2) To address MPSoCs without temperature sensors, we utilize predictors to estimate future workload dynamics, such as core utilization and instructions per cycle (IPC), and demonstrate the differences with temperature prediction.
3) In addition to the eight-core multithreaded UltraSPARC T1 (with real-life workloads as measured by the continuous system telemetry harness (CSTH) [13]), we evaluate our proactive thermal management policy on an architecture-level simulator configured for a 16-core single-threaded MPSoC. This paper provides comparisons between multi- and single-threaded systems, discusses the scalability of the policies, and analyzes various proactive and reactive thermal management methods.

The rest of this paper starts with an overview of the related work. Section III discusses the ARMA predictor and the online adaptation framework. We compare ARMA with other predictors in Section IV. Section V demonstrates how to predict workload dynamics for MPSoCs without thermal sensors. In Section VI, we explain the thermal management techniques we study in this paper. Section VII provides the methodology and results, and we conclude in Section VIII.

II. Related Work

In this section, we discuss the techniques for multicore scheduling and thermal management. We also briefly investigate previous work on energy management, as energy consumption affects temperature significantly.

A number of MPSoC scheduling techniques with performance and energy objectives have been introduced previously (e.g., [25]). Minimizing energy on MPSoCs using dynamic voltage-frequency scaling (DVS) has been formulated using a two-phase framework in [36]. As power-aware policies are not always sufficient to prevent temperature-induced problems, thermal modeling and management methods have been proposed. HotSpot [28] is an automated thermal model to calculate transient temperature response given the physical characteristics and power consumption traces. A fast thermal emulation framework is introduced in [4], which reduces the simulation time considerably while maintaining accuracy. Including temperature as a constraint in the cosynthesis framework and in task allocation for platform-based system design is introduced in [15]. Reliability-aware microprocessor [30] provides a reliability model for temperature-related failures and optimizes the architectural configuration and power/thermal management policies for reliable design. In [24], it is shown that aggressive power management can adversely affect reliability due to fast thermal cycles, and the authors propose a policy optimization method for MPSoCs, which saves power while meeting reliability constraints. A hardware–software emulation framework for reliability analysis is proposed in [3], and a reliability-aware register assignment policy is introduced as a case study.

Dynamic thermal management controls overheating by keeping the temperature below a critical threshold. Computation migration and fetch toggling are examples of such techniques [28]. Heat-and-run performs temperature-aware thread assignment and migration for multicore multithreaded systems [12]. Kumar *et al.* propose a hybrid method that coordinates clock gating and software thermal management techniques such as temperature-aware priority management [20]. The multicore thermal management method introduced in [11] combines distributed DVS with process migration. The temperature-aware task scheduling method proposed in [9] achieves better thermal profiles than conventional thermal management techniques without introducing a noticeable impact on performance.

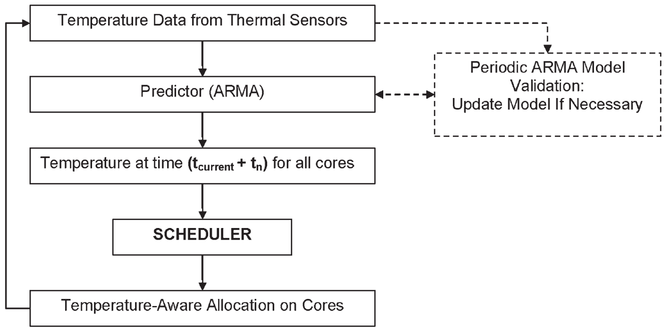
Temperature Data from Thermal Sensors
Periodic ARMA Model Validation: Update Model If Necessary
Predictor (ARMA)
Temperature at time ($t_{current}$ + t_n) for all cores
SCHEDULER
Temperature-Aware Allocation on Cores

Fig. 1. Flowchart of the proposed technique.

In this paper, we introduce a proactive temperature management method for MPSoCs, which can adapt to dynamic changes in system behavior. The key difference from reactive management is that, as opposed to taking action after the temperature reaches a certain level, our technique estimates the hot spots and temperature variations in advance and modifies the job allocation to minimize the adverse effects of temperature. For systems without temperature sensors, we show how to predict workload dynamics instead of temperature and discuss how to perform proactive management based on workload predictions. We also extensively compare various reactive and proactive thermal management techniques (i.e., in terms of their thermal behavior and performance) not only in simulation but also on a real system implementation.

III. Prediction With ARMA Models

A. ARMA Predictors

In this section, we provide an overview of our proactive temperature management approach and explain the methodology for accurate temperature prediction at runtime. Fig. 1 shows an overview of our technique. Based on the temperature observed through thermal sensors, we predict temperature t_n steps into the future using an ARMA model. Utilizing these predictions, the scheduler then allocates the threads to cores to balance the temperature distribution across the die. The ARMA model utilized for temperature forecasting is derived based on a temperature trace representative of the thermal characteristics of the current workload. During execution, the workload dynamics might change, and the ARMA model may no longer be able to predict accurately. To provide runtime adaptation, we monitor the workload through the temperature measurements, validate the ARMA model, and update the model if needed. The online adaptation method is explained in Section III-B.

ARMA models are mathematical models of autocorrelation in a time series. In this paper, we use ARMA models to predict the future temperature of cores using the observed temperature values in the past. ARMA model assumes that the modeled process is a stationary stochastic process and that there is a serial correlation in the data. In a stationary process, the probability distribution does not change over time, and the mean and variance are stable. Based on the observation that workload characteristics are correlated during short time windows, and that temperature changes slowly due to thermal time constants, we assume that the underlying data for the ARMA model are stationary. We adapt the model when the ARMA model no longer fits the workload. Thus, the stationary assumption does not introduce inaccuracy.

$$y_t + \sum_{i=1}^{p}(a_i\, y_{t-i}) = e_t + \sum_{i=1}^{q}(c_i\, e_{t-i}). \tag{1}$$

An ARMA(p, q) model is described by (1). In the equation, y_t is the value of the series at time t (i.e., temperature at time t), a_i is the lag-i autoregressive (AR) coefficient, c_i is the moving average (MA) coefficient, and e_t is called the noise, error, or residual. The residuals are assumed to be random in time (i.e., not autocorrelated) and normally distributed. p and q represent the orders of the AR and MA parts of the model, respectively.

ARMA modeling has two steps: 1) *identification and estimation*, which consist of specifying the order and computing the coefficients of the model (coefficients are computed by software with little user interaction) and 2) *checking the model*, where it is ensured that the residuals of the model are random and the estimated parameters are statistically significant.

1) Identification and Estimation: During identification, we use an automated trial-and-error strategy. We start by fitting the training data with the simplest model, i.e., ARMA$(1, 0)$, measure the "goodness of fit," and increase the order of the model if the desired fit is not achieved. At each iteration, to fit the data with the current order of ARMA model, coefficients are computed using a least-squares fit. Other methods can be utilized for coefficient estimation.

We use the *final prediction error* (*FPE*) [22] to evaluate the goodness of fit of the models. Once the FPE is below a predetermined threshold, we halt the trial-and-error loop. FPE is a function of the residuals and the number of estimated parameters. As FPE takes the number of estimated parameters into account, it compensates for the artificial improvement in fit that could come from increasing the order of the model. The FPE is given in the following, where V is the variance of model residuals, N is the length of the time series, and $n = p + q$ is the number of estimated parameters in the model:

$$FPE = \frac{1 + n/N}{1 - n/N} \cdot V. \tag{2}$$

2) Checking the Model: For checking that the model residuals are random, or uncorrelated in time, we look at the *autocorrelation function* (*ACF*). Autocorrelation is the cross correlation of a signal with itself as a function of lag time and is useful for finding repeating patterns in a signal if there are any. If model residuals are random, the ACF of all residuals (except for lag zero) should fluctuate close to zero. The residuals are assumed as random if the ACF for the majority of the trace is in between the predetermined confidence intervals.

As an example, we have applied the ARMA prediction methodology to a sample temperature trace. The trace is obtained through HotSpot [28] for a Web server workload running on a system with a thermal management policy that swaps workload among hot and cold cores periodically, causing thermal cycles. We show a part of the trace in Fig. 2, while the total length of the example trace is 200 samples long, sampled at every 100 ms. Using the first 150 samples of the data as the training set and $FPE \ll 1$, we formed an ARMA$(5, 0)$ model. It should be noted that, for most of the real-life workloads, we experimented with much shorter training sets (i.e., 20–50 samples) that were sufficient for forming an ARMA model with the desired fit.

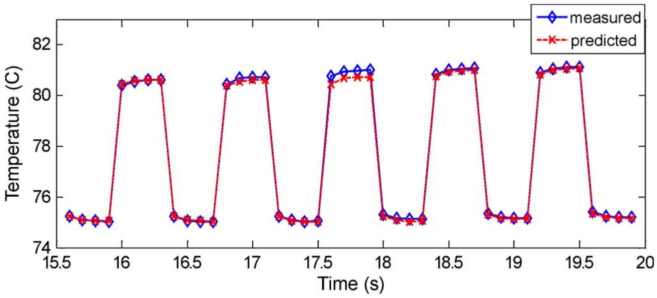

Fig. 2. Temperature prediction.

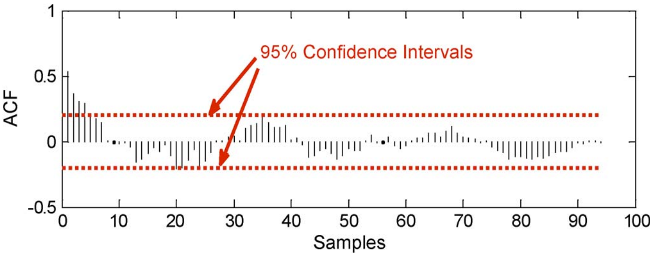

Fig. 3. ACF of the residuals.

We saved the last 50 samples of the data to test our prediction method. We used the ARMA model to predict five steps (i.e., 500 ms) into the future. Fig. 2 shows that the prediction matches the observed values closely. For temperature curves with less temporal variation, designing an accurate ARMA predictor is even easier. Fig. 3 shows the ACF of the residuals for the example in Fig. 2. Each sample refers to a 100-ms interval. The horizontal lines show the 95% confidence intervals. In our automated methodology, we observe the percentage of ACF values within the 95% confidence interval. If most of the ACF values fall within the 95% range, we declare that the residuals are random.

Computing the ARMA model has relatively low overhead. For example, in Matlab, an ARMA$(p, 0)$ model with $p \leq 10$ (no MA component) for a training data set of 50 samples can be computed in less than 150 ms, and an ARMA(p, q) model up to the fifth order can be computed in less than 300 ms. The computation and validation of the model together take between 250 and 500 ms. Note that implementing the ARMA process in C/C++ and optimizing the source code would significantly reduce the overhead.

Note that, even though each core's ARMA model considers solely the temperature behavior of that particular core, the temperature trace of a core inherently takes into account the thermal behavior of the neighbor cores. This is due to the fact that temperature is not only dependent on the power consumption of a unit, but also on the floorplan and the power/thermal characteristics of other units on the die. While it is possible to develop prediction techniques that consider a joint power/temperature profile of a set of units for forecasting, in our experiments, we observed that considering each core's thermal trace individually results in accurate predictions.

B. Online Adaptation

ARMA models are accurate predictors when the time series data are stationary. Since the workload dynamics vary at runtime, the temperature characteristics may diverge from the training data we used for forming the initial ARMA model. In order to adapt to changes in the workload, we propose monitoring the temperature dynamics and validating the ARMA model. When we determine that the current workload deviates from the initial assumptions used for forming the ARMA model, we update the model on the fly.

We use SPRT to detect changes over time in the statistical characteristics of the residual signals. Applying SPRT on the residuals provides the earliest possible indication of anomalies [34], where anomaly is defined as the residuals drifting from their expected distribution. Instead of using a simple threshold value for detection (e.g., setting a threshold for the standard deviation of the prediction error), SPRT performs statistical hypothesis tests on the mean and variance of the residuals. These tests are conducted on the basis of the user-specified false- and missed-alarm probabilities of the detection process, allowing the user to control the likelihood of the missed detection of residual drifts or false alarms.

To perform online validation, we maintain a history window of temperature on each core. The window length is empirically selected based on thermal time constants and workload characteristics. To monitor the prediction capabilities of the model at runtime, for each new data sample, we compute the residual by differencing the predicted data from the observed data. Our goal at runtime is to detect if there is a *drift* in residuals, where a drift refers to the mean of residuals moving away from zero (recall that, for an ARMA model with good prediction capabilities, the residuals should fluctuate close to zero). Detecting the drift quickly is important for maintaining the accuracy of the predictor, as such a drift shows that the model no longer fits the current temperature dynamics.

Specifically, we declare a drift when the sequence of the observed residuals appears to be distributed about mean $+M$ or $-M$ instead of around zero, where M is our preassigned system disturbance magnitude. A typical value for M would be $(3 * \sqrt{V})$, where V is the variance of the residuals in the training data set.

At time instant t, the residuals (R) can be computed by (3), where $T'_i(t)$ is the prediction and $T_i(t)$ is the measurement.

$$R(t) = T_i(t) - T'_i(t). \tag{3}$$

SPRT then decides between the following two hypotheses.

1) H_1: $R(t)$ is drawn from a probability density function (pdf) with mean M and variance σ^2.
2) H_2: $R(t)$ is from a pdf with mean zero and variance σ^2.

In other words, we detect that there is a drift if SPRT decides on H_1. If H_1 or H_2 is true, we wish to decide on the correct hypothesis with probability $(1-\beta)$ or $(1-\alpha)$, respectively, where α and β are the false- and missed-alarm probabilities. Small values such as 0.01 or 0.001 are used for α and β.

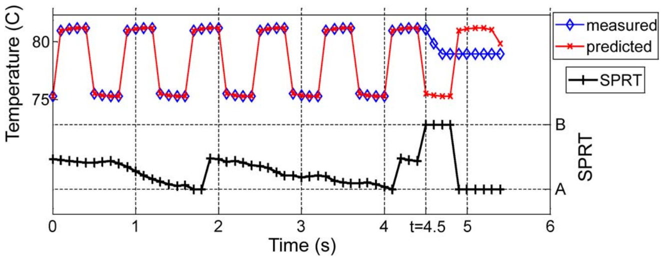

Fig. 4. Online detection of variations in thermal characteristics.

SPRT is applied to detect the drift (i.e., anomaly) in residuals by computing the log likelihood ratio in (4), where $p(./H_2)$ is the joint density function assuming no fault, $p(./H_1)$ is the joint density function assuming fault, and N is the number of observations.

$$LR_N = \ln \frac{p\left[R(1), R(2), \ldots, R(N)/H_1\right]}{p\left[R(1), R(2), \ldots, R(N)/H_2\right]}. \tag{4}$$

If $LR_N \geq B$, we accept H_1, meaning that the residuals show significant change from the assumptions, and if $LR_N \leq A$, we accept H_2. If one of the hypotheses is accepted, the SPRT computation is restarted from the current sample. Otherwise (i.e., $A < LR_N < B$), we continue the measurements. The bounds A and B are defined as in

$$A = \ln\left(\frac{\beta}{1-\alpha}\right) \qquad B = \ln\left(\frac{1-\beta}{\alpha}\right). \tag{5}$$

Following the derivation provided in [14], the value of SPRT can be represented as in (6). In the equation, M is the system disturbance magnitude as defined previously, and σ^2 is the variance of the residuals in the training set

$$SPRT = \frac{M}{\sigma^2} \sum_{i=1}^{N} \left(R(i) - \frac{M}{2}\right). \tag{6}$$

Note that the M and σ^2 values are computed at the beginning and then fixed until the ARMA model is updated. Thus, at runtime, during each sampling interval, the SPRT equation effectively performs one addition and one multiplication. Because of the simplicity of computation shown in (6), the cost of computing SPRT after each observation is very low (negligible in our measurements). Moreover, as shown in [34], there is no other procedure that has the same error probabilities with shorter average sampling time than SPRT. We have picked SPRT as the online monitoring tool in this paper due to both its guarantee for fast detection of changes and low computation overhead.

In Fig. 4, we demonstrate a case where the temperature dynamics change and the SPRT detects this change immediately (see $t = 4.5$ s in the figure). A and B correspond to the SPRT thresholds of ±6.9068 for α and β values of 0.001. When $SPRT >= 6.9068$ (i.e., $LR_N \geq B$), we declare that the residuals have a drift from the training data, initiating the computation of a *new* ARMA model. Recall that, when $SPRT <= -6.9068$ (i.e., $LR_N \leq A$), we accept the hypothesis that the mean of the residuals is zero. In both cases, the SPRT computation is restarted.

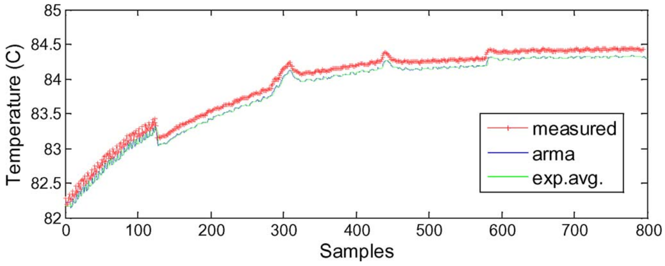

Fig. 5. Comparison of predictors—stable temperature.

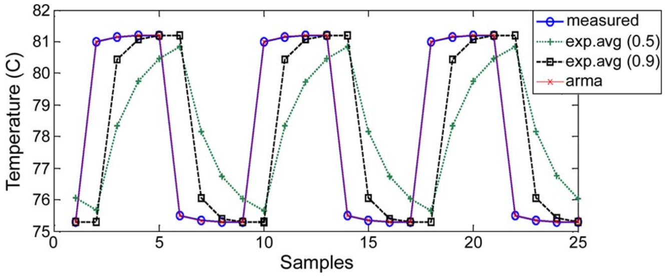

Fig. 6. Comparison of predictors—thermal cycling.

We also compared the SPRT detection with monitoring the standard deviation of the residuals. The prediction capability of an ARMA model can be examined by computing the standard deviation of the prediction error. If the dynamic characteristics of the temperature time series can be well represented by the model, the standard deviation of the associated prediction error should be relatively small. It is generally recommended to keep the standard deviation of prediction errors to less than 10% of the standard deviation of the original signal. This condition implies that the ARMA model is able to capture more than 90% of the underlying dynamics of the system. Using a 10% threshold, the standard deviation method can quickly detect the change in temperature dynamics in the case of abrupt changes such as in Fig. 4. However, for gradual shift in thermal dynamics, it may fail to capture the drift immediately. SPRT guarantees the fastest detection for the given false- and missed-alarm probabilities.

IV. Comparing Predictors

In this section, we compare ARMA with various predictors in terms of their prediction and adaptation capabilities, and their computation and hardware overhead.

A. Exponential Averaging

A well-known method for prediction is exponential moving averaging. In Figs. 5 and 6, we compare ARMA prediction with exponential average prediction for an execution slice of a highly utilized Web server workload and the previous trace used in Fig. 2, respectively. The exponential average predictor estimates the current value of the series as $y_t = \alpha T_{t-1} + (1 - \alpha) y_{t-1}$, where y_t is the predicted temperature (i.e., exponential average) at time t, T_{t-1} is the measured temperature at time $t-1$, and α is a constant $(0 \leq \alpha \leq 1)$. We used $\alpha = 0.9$ in Fig. 5 and $\alpha = \{0.5, 0.9\}$ in Fig. 6.

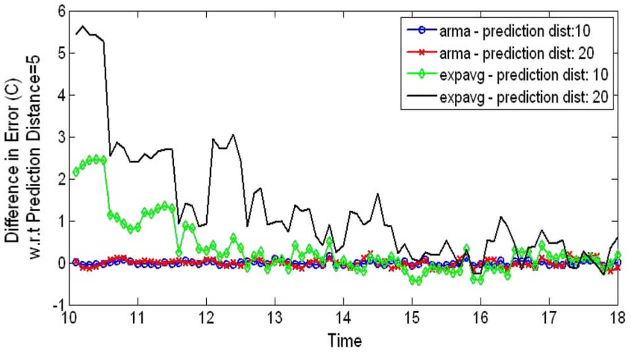

Fig. 7. Predicting further ahead with exponential averaging.

When we have a relatively stable temperature, an exponential average predictor works well, providing almost the same values as the ARMA predictor in Fig. 5. However, when there are rapid temperature changes, such as thermal cycling, the exponential average predictor performs poorly, such as in Fig. 6. In addition, even though the exponential average predictors with $\alpha = 0.9$ and $\alpha = 0.5$ perform very similarly in the first example, there is a significant effect of the α value in the thermal cycling case, which would require the user to determine α accordingly. Contrarily, ARMA predictor has an automated process of forming the model with high accuracy. The overhead of evaluating the ARMA or the exponential average model at runtime is very similar, as both models only compute a polynomial equation for each sample.

The ideal number of steps to predict ahead depends on the system and workload characteristics. In our experiments, we predict 500 ms (i.e., five steps) into the future, which provided good results for our proactive thermal management policies. However, for different system and workload characteristics, the preferred prediction distance may vary. For example, for systems with less variant workload, predicting further ahead and using a lower sampling rate for polling the temperature sensors can be sufficient.

For the experiment in Fig. 7, we increased the prediction distance to 10 and 20 steps to evaluate the accuracy of ARMA and exponential averaging predictors as a function of the prediction distance. When the goal is forecasting several time steps into the future, the prediction accuracy of exponential averaging degrades significantly. For this experiment, we used a 200-sample temperature trace for a CPU-bound SPEC 2000 suite workload, where the temperature was changing within a 1.5°C range. The plot shows the difference of error (in degree Celsius) in comparison to predicting five steps into the future with the same predictor. While the ARMA predictor's accuracy is stable, the error margin of the exponential predictor increases considerably when predicting ahead.

B. History Predictor

In Section III, we showed that it is possible to predict the future temperature accurately based on the previous thermal measurements. Following this insight, we built a history predictor, which is similar to a global branch predictor and consists of a shift register that tracks the last few observed values. The length of the history is specified by the shift register depth. At each sampling period, the register is updated with the last

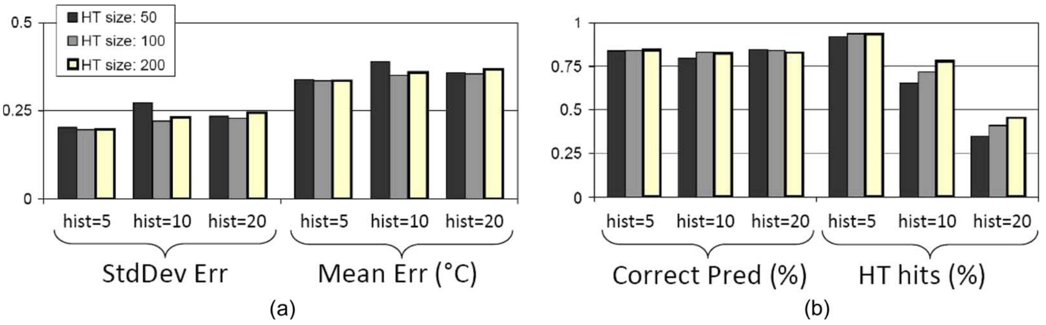

Fig. 8. Accuracy–size tradeoff for the history predictor.

measurement. This updated shift register content is used to index a history table (HT). The HT holds several previously observed thermal patterns, with their corresponding next value predictions. The shift register index is associatively compared to the stored valid HT tags, and if a match is found, the corresponding HT prediction is used as the final prediction. We keep an invalid entry for each tag to track the ages of different HT tags for applying a least recently used replacement policy when the HT is full. This predictor is similar to the global history predictor used for predicting power phases in [17]. When the shift register does not hit the HT, the predictor behaves like a last-value predictor and assumes that the future temperature value will be the same as the last observed temperature.

One issue with the history predictor is the precision of temperature data. We have performed experiments where we stored temperature readings with one or two decimal places. However, even for relatively stable temperature profiles, obtaining a reasonable percentage of hits on the HT was not possible when we considered the decimal places. In addition, even when we maintain one decimal digit, the required HT size for predicting with high accuracy becomes considerably large (i.e., we would have to have new entries in the table to accommodate even slight changes in the decimal digit). For this reason, for the history predictor, we round the temperature measurements to the nearest integer values and only predict temperature in integers.

In Fig. 8(a) and (b), we show the accuracy for various HT sizes and history lengths. In this experiment, we have used the same temperature trace we used for Fig. 7 and predicted five steps ahead. In Fig. 8(a), we compare the standard deviation of error and the mean error (in degree Celsius) for the prediction, where error is the difference of the measured trace and predictions. For this workload, we observe that increasing the history length does not bring much benefit; however, increasing the table size reduces the magnitude of errors. In Fig. 8(b), we demonstrate the correct prediction ratio (with respect to the integer temperature trace) and the hit rate for the HT. While increasing the history length reduces the hit rate as expected, the accuracy does not get affected by this. This is due to the fact that, for stable profiles, the last-value predictor compensates well when the history predictor cannot predict. Note that increasing the table size over 100 does not bring additional benefits, which motivates the use of a small-size table to achieve enough accuracy with lower hardware overhead.

Fig. 9 shows the ARMA predictor and the history predictor (with an HT size of 100 and a history length of five) for predicting five steps ahead (i.e., 500 ms). We observe that the ARMA captures the thermal dynamics almost exactly, while the history predictor can predict the integer value of the temperature with reasonable accuracy. For the repeating patterns of workload, such as several applications being time-multiplexed on a core, and stable thermal profiles, the history predictor can predict with high accuracy and does not require a training phase (except for the first time an application is run), provided that the HT is large enough to maintain the entries associated with all of the applications.

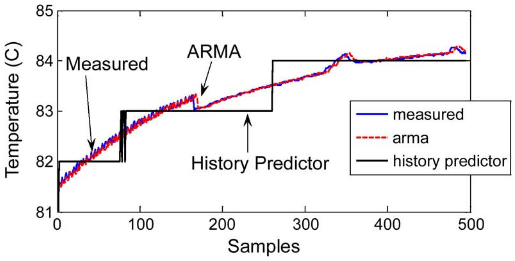

Fig. 9. Comparison of ARMA and history predictor.

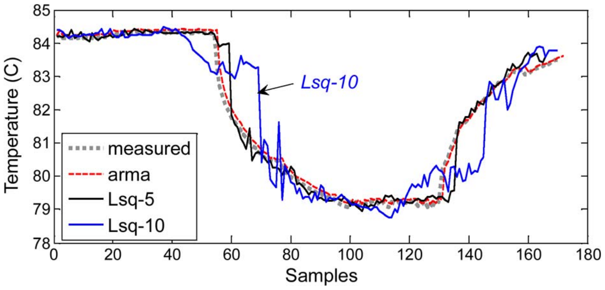

Fig. 10. Comparison of ARMA and recursive least squares predictor.

C. Recursive Least Squares

Another recently proposed temperature prediction method is using recursive least squares [35]. In Fig. 10, we compare the prediction accuracy of the least squares approach with ARMA. We trained both predictors with 50 samples of the temperature data. While least squares method with a prediction distance of five (shown as $Lsq - 5$) has similar accuracy as ARMA ($prediction\ distance = 10$), its accuracy drops rapidly when we increase the forecasting distance (t_n) to ten steps ($Lsq - 10$). This trend continues even more dramatically for higher t_n.

Note that both of the aforementioned methods can predict the data in the history window they are trained with the desired degree of accuracy. If one keeps adding enough terms, it is even

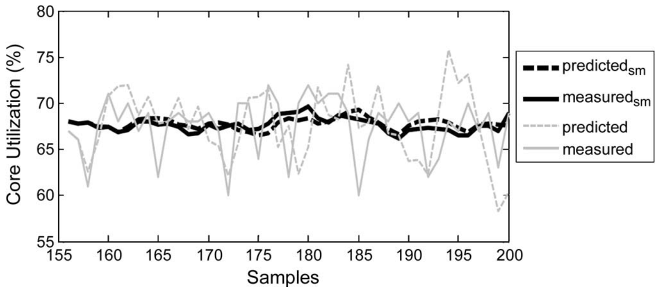

Fig. 11. Prediction of core utilization.

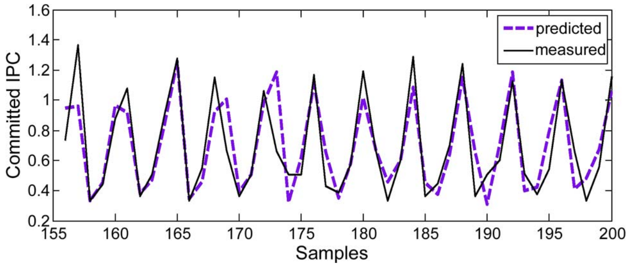

Fig. 12. Prediction of committed IPC.

possible to fit through every single observation in the history window. However, typically, we would not want to do that because, usually, there is random measurement noise on the time series and there is no value to learning the noise. Thus, the differentiating point of recursive least-squares and ARMA arises when we are forecasting further into the future. As we increase t_n, recursive least-squares does significantly worse than ARMA. The reason is that, as soon as you predict more than a few time steps into the future, the term with the biggest exponent in the least-squares fitting function dominates, and the prediction accuracy degrades from that point on.

Another important advantage of ARMA in comparison to least-squares is in the overhead. Recursive least-squares method continuously updates the coefficients of the model as new data arrive (otherwise, accuracy drops), whereas the SPRT support enables us to update the model only when it is necessary. In addition, the length of the polynomial in the least-squares estimation needs to be set manually, which can unnecessarily increase computation overhead if set to a larger value than needed. On the other hand, we use an automated and fast trial-and-error strategy for setting the number of terms in the ARMA model.

V. Workload Prediction

The predictors discussed previously assume a telemetry infrastructure on the chip, which provides temperature measurements at the desired granularity. In a number of systems, we may not have a thermal sensor for each core or sensors may degrade and fail during the system lifetime. To apply a proactive management strategy for such cases, in this section, we discuss how the workload parameters can be predicted.

For workload prediction, we demonstrate the prediction of two parameters: 1) core utilization and 2) the IPC of committed instructions. Core utilization is a good measure of how busy the core is and hence provides an insight for the power consumption, particularly in multithreaded systems, where we may not have access to measuring per-thread IPC. For single-threaded systems, IPC tends to have a strong correlation with the power consumption. While such performance metrics may not directly reflect the thermal behavior of cores, they still provide an estimation of whether the power consumption is increasing or decreasing in the near future. Therefore, the forecast of future workload can be utilized to perform proactive temperature management, assuming a correlation between high utilization/IPC and high power consumption.

Figs. 11 and 12 show the traces of core utilization and committed IPC, respectively, and the prediction results obtained by ARMA. The core utilization results are collected for medium utilized Web application on a multithreaded system. The IPC trace belongs to `bzip` running on a single-threaded architecture. Both predictors were trained using 150 samples, and the prediction was performed for the following 50 samples. Note that the workload parameters may have short-term spikes due to changing application characteristics, while these do not typically get reflected in the temperature response due to the thermal time constants. This is particularly the case for core utilization. To achieve more accurate prediction for core utilization, we applied a smoothing function (i.e., moving averaging) to the workload traces. The smoothed-out utilization and prediction signals are demonstrated with the subscripts sm in Fig. 11. For the original trace, the accuracy of utilization prediction is significantly lower than the temperature prediction. However, when the data are smoothed out first, the predictor works more accurately.

Even though bzip is a highly IPC-variant benchmark, Fig. 12 shows that IPC can be predicted with high accuracy. Note that applications typically have different phases of performance, and SPRT would detect such a change immediately. The substantial accuracy difference between predicting IPC and core utilization is mainly due to the difference between observing a single thread and observing multiple threads at the same time. The core utilization results are collected on a multithreaded system, where the core is running a set of threads rather than a single application.

VI. Proactive Temperature Management

We study various thermal management techniques for MPSoCs and propose a proactive temperature-aware job allocation technique. This section discusses the details of all the techniques we implemented. We consider both single-threaded and multithreaded systems in this paper. In the system model for the multithreaded systems, each core has a *dispatching queue*, which holds the threads allocated to that core. This is the typical abstraction in modern multicore OS schedulers based on multilevel queuing. The dispatcher allocates the incoming threads to queues based on the current policy.

The default policy we evaluate (i.e., default policy in modern OSs, e.g., Solaris) is `Dynamic Load Balancing`, which assigns an incoming thread to the core it ran previously, if the thread ran recently. If the thread has not run recently, then the dispatcher assigns it to the core that has the lowest priority thread in the queue. The dispatcher first tries to assign the thread based on locality (e.g., if several cores are sharing a cache or on the same chip, etc.) if possible. At runtime, if there is a significant imbalance among the queues, the threads are migrated to have more balanced utilization.

Many temperature management methods rely on dynamic temperature data acquired from the system; thus, we assume that each core has a temperature sensor. Current chips typically contain several sensors, and these sensors can be read by the CSTH for collecting and analyzing time series sensor data [13]. The management policies observe the system characteristics at regular intervals (i.e., ticks) to make decisions.

1) Power Management: Many current MPSoCs have power management capabilities to reduce the energy consumption. Even though the power management techniques do not directly address temperature, they affect the thermal behavior significantly. We implement two commonly used power management methods: DPM and DVS. For DPM, we utilize a fixed time-out policy, which puts a core to sleep state if it has been idle longer than the time-out period. We set the time-out as the breakeven time [19]. The DVS policy observes the core utilization over a given length of recent history and reduces the frequency/voltage proportionally.

2) Reactive Thermal Management: Several reactive thermal management techniques have been proposed in literature (e.g., [12]). In this paper, we implement some of the most commonly used methods.

Reactive thread Migration (*R-Mig*) migrates the workload from a core if the core's temperature is above the threshold to the coolest core available. In single-threaded systems, this corresponds to migrating the currently running job or swapping the jobs among the hot and cool cores. In multithreaded environment, the technique migrates the current threads in the hot core's dispatch queue to other cool cores or swaps threads among hot and cool cores.

Reactive DVS (*R-DVS*) reduces the voltage/frequency (V/f) setting on a core if the threshold temperature is exceeded, which is similar to the frequency scaling approach in [28]. We assume three built-in V/f states in our experiments. The policy continues to reduce the (V/f) level at every tick as long as the temperature is above the threshold. When the temperature is below the critical threshold, then the V/f setting is increased.

3) Proactive Thermal Management: The proactive methods utilize the temperature prediction introduced in Section III-A. The motivation behind proactive management is to avoid thermal emergencies before they occur and, thus, to minimize the adverse effects of hot spots and temperature variations at lower performance cost.

In the workload allocation techniques we propose, we do not change the priority assignment of the threads or the time slices allocated for each priority level. This paper focuses on finding effective dispatching methods to reduce temperature-induced problems without affecting performance.

Proactive thread Migration (*P-Mig*) moves workload from cores that are projected to be hot in the near future to cool cores. *Proactive DVS (P-DVS)* reduces the V/f setting on a core if the temperature is expected to exceed the critical threshold. These two policies are the same as their reactive counterparts, except that they get triggered by the temperature estimates instead of the current temperature.

Proactive temperature balancing (PTB) follows the principle of locality (i.e., allocating the threads on the same core they ran before) during the initial assignment as in the default policy. At every scheduler tick, if the temperatures of cores are predicted to have imbalance in the next interval, threads waiting on the queues of potentially hotter cores are moved to cooler cores. This way, the thermal hot spots can be avoided, and the gradients are prevented by thermal balancing.

In a single-threaded system, we bound the number of migrations to avoid the unnecessary performance cost. The migration of the jobs on all the hot cores can cause thermal oscillations. We start performing migrations from the hottest core and migrate only if the workload on the hot core's neighbors has not been migrated during the current tick. Note that, in a multithreaded environment, threads waiting in the queue are moved unless the threshold is already exceeded; thus, migration does not stall the running thread. This is in contrast to moving the actively running threads in thread migration policies discussed earlier. As moving the waiting threads in the queues is already performed by the default policy for load balancing purposes, this technique does not introduce additional overhead. In PTB for multithreaded systems, the number of threads to migrate is proportional to the spatial temperature difference among the hot and cool cores.

VII. Experimental Results

In this section, we evaluate the thermal management techniques discussed in the previous section. In the results, DLB is the default load balancing policy, R-Mig (P-Mig) and R-DVS (P-DVS) refer to the reactive (proactive) migration and voltage scaling, respectively, and PTB is the proposed job allocation policy (i.e., combined with ARMA predictor). All predictors in this section predict five steps ahead (i.e., 500 ms, assuming a 100-ms sampling rate).

We show two sets of experimental results. The first set is based on the UltraSPARC T1 processor [21]. In the second set of results, we use an architecture-level simulation framework to simulate performance, power, and temperature and provide results on a hypothetical high-performance 16-core architecture manufactured at 65 nm.

The threshold temperature for the management policies is 85 °C, which is considered a high temperature for our system. Higher temperature increases the failure rates, for example, for electromigration related failures, an increase in temperature from 80 °C to 85 °C reduces the mean time to failure by 30% [18]. In this section, the hot spot results demonstrate the percentage of "time spent above 85 °C." The spatial gradient results summarize the percentage of time that gradients above 15 °C occur, as gradients of 15 °C–20 °C start causing clock skew and delay issues [1]. The spatial distribution is calculated by evaluating the temperature difference between hottest and coolest cores at each sampling interval. For metallic structures, assuming the same frequency of thermal cycles, when ΔT increases from 10 °C to 20 °C, failures happen 16 times more frequently [18]. Thus, we report the temporal fluctuations of magnitude above 20 °C. The ΔT values we report are computed over a sliding temperature history window (i.e., maximum ΔT in the history window) and averaged over all cores.

A. UltraSPARC T1 Implementation

The first set of experimental results is based on the UltraSPARC T1 [21]. The experimental flow consists of gathering workload traces, applying policies (scheduling, DVS, etc.) on the given workload, computing the corresponding power traces, and finally calculating the temperature response.

The results marked as *real implementation* refer to our implementation of the policies in Solaris, where we ran the workload on the UltraSPARC T1 in real time. Some of our policies utilize temperature readings from all cores, and UltraSPARC T1 does not contain a sensor for each core. To obtain detailed thermal data in synchronization with the scheduling experiments, using a shared file, we piped the utilization data collected from the target machine to another computer that was running the thermal simulator. The utilization data were converted into the equivalent power trace by the thermal simulator, and the temperature results were computed for the next interval and then were passed back to the target system (which was running the thermal management policies). A separate computer in the private network was assigned to run the thermal simulator to avoid interfering with the workload dynamics on the target system. In the *real implementation*, the core utilization statistics were passed to the thermal simulator at every 1-s interval, and the thermal simulations were sampled at every 100 ms, which provided good precision. To implement thread migration, we utilized the existing migration routine in the OS dispatcher and included additional temperature-induced triggers accordingly.

The results marked as *simulator* are from our simulation infrastructure attached to the power/thermal model, where we used the real-life workload traces again, but this time implemented the scheduling policies within the simulator that is a replica of the multicore system model. Again, the temperature sampling rate was set at 100 ms.

We leveraged the CSTH [13] to gather detailed workload characteristics of real applications. We sampled the utilization percentage for each hardware thread at every second using `mpstat` [23]. We recorded half-an-hour-long traces for each benchmark. To determine the active/idle time slots of cores, we recorded the length of user and kernel threads using `DTrace` [23].

We ran the following sets of benchmarks: 1) Web server; 2) database; 3) common integer; and 4) multimedia. To generate a Web server workload, we ran SLAMD [29] with 20 and 40 threads per client to achieve medium and high utilizations, respectively. For database applications, we tested MySQL using `sysbench` for a table with one million rows and 100 threads. We also ran compiler (`gcc`) and compression/decompression (`gzip`) benchmarks. For multimedia, we ran mplayer (integer) with a 640 × 272 video file. Several instances of the benchmarks were executed simultaneously for the integer and multimedia benchmarks to achieve a reasonable utilization on the MPSoC. We summarize the details of our benchmarks in Table I. Utilization ratios are averaged over all cores throughout the execution. Using `cpustat`, we also recorded the cache misses and floating point (FP) instructions per 100 thousand instructions.

The peak power consumption of SPARC is similar to its average power [21]; therefore, we assumed that the instantaneous power consumption is equal to the average power at each state (active, idle, and sleep). The average power consumption for UltraSPARC T1 (including leakage) and the area distribution of the units are provided in Table II, and the floorplan is available in [21].

TABLE I
WORKLOAD CHARACTERISTICS

	Benchmark	Avg Util (%)	L2 I-Miss	L2 D-Miss	FP instr
1	Web-med	53.12	12.9	167.7	31.2
2	Web-high	92.87	67.6	288.7	31.2
3	Database	17.75	6.5	102.3	5.9
4	Web & DB	75.12	21.5	115.3	24.1
5	gcc	15.25	31.7	96.2	18.1
6	gzip	9	2	57	0.2
7	MPlayer	6.5	9.6	136	1
8	MPlayer&Web	26.62	9.1	66.8	29.9

TABLE II
POWER AND AREA DISTRIBUTIONS OF THE UNITS

Component Type	Power (%)	Area (%)
Cores	65.27	37.66
Caches	25.50	50.69
Crossbar	6.01	5.84
Other	3.22	5.81

We estimated the power at lower voltage levels based on the equation $P \propto f * V^2$. We assumed three built-in voltage/frequency settings in our simulations. To account for the leakage power variation at runtime, we used the second-order polynomial model proposed in [31]. We determined the coefficients in the model empirically to match the normalized leakage values in [31]. As we know the amount of leakage at the default voltage level for each core, we scaled it based on this model for each voltage level, considering the temperature change as well. We used a sleep state power of 0.02 W, which is estimated based on the sleep power of similar cores. To compute the power consumption of the crossbar, we scaled the power according to the number of active cores and the memory access statistics.

We used HotSpot version 4.2 [28] as the thermal modeling tool and modified the floorplan and thermal package characteristics for UltraSPARC T1. We initialized HotSpot with steady-state temperature values.

First, we provide the *simulator* results. Table III shows a detailed analysis of the hot spots observed on the system for each workload and also the average performance results. We show the percentage of time spent above 85 °C for all the workloads, and also the average results for the cases with no power management and DPM. The performance results shown in the table are normalized with respect to the default policy's performance. We computed the performance based on the average delay we observed in the thread completion times. The reactive migration of workload or applying temperature triggered DVS cannot eliminate all the hot spots, particularly for workloads with medium to high utilization level. Performing migration or DVS proactively achieves significantly better results while also reducing the performance cost. The cost is lower with the proactive approaches as they maintain the temperature at lower levels, requiring fewer overall number of migrations or shorter periods of DVS. Note that, once a temperature threshold is reached, execution at the default speed is not allowed on a core until the temperature is lowered. Moreover, when a system is highly utilized, swapping threads may not reduce the temperature sufficiently, and frequent threshold triggers may occur as new threads arrive.

TABLE III
THERMAL HOT SPOTS AND PERFORMANCE (*simulator*)

	no PM						DPM					
Workload	DLB	R-Mig	P-Mig	R-DVS	P-DVS	**PTB**	DLB	R-Mig	P-Mig	R-DVS	P-DVS	**PTB**
Web-med	25.9	12.9	5.9	7.7	3.3	3.8	19.5	10.9	3.4	4.6	2.0	2.5
Web-high	39.1	22.1	13.3	19.2	10.4	10.6	37.4	21.6	10.7	14.8	8.4	8.5
Database	8.3	2.1	1.2	1.5	1.1	1.0	4.6	1.5	0.0	1.1	0.0	0.0
Web&DB	32.4	15.3	7.1	10.7	5.2	4.8	27.8	13.2	7.7	6.7	4.8	4.6
gcc	7.2	1.8	1.5	0.5	1.3	0.7	3.8	1.3	0.0	0.1	0.0	0.0
gzip	2.9	0.6	0.0	0.1	0.0	0.0	1.3	0.5	0.0	0.0	0.0	0.0
Mplayer	4.9	0.7	0.0	0.4	0.0	0.0	1.7	0.5	0.0	0.1	0.0	0.0
Mplayer&Web	13.3	9.4	5.3	4.9	2.4	2.1	8.9	7.2	5.2	4.1	1.2	1.1
AVG	16.8	8.1	4.3	5.6	3.0	2.9	13.1	7.1	3.4	3.9	2.1	2.1
AVG Perf.	1.00	0.96	0.97	0.89	0.91	0.98	1.00	0.95	0.96	0.87	0.90	0.97

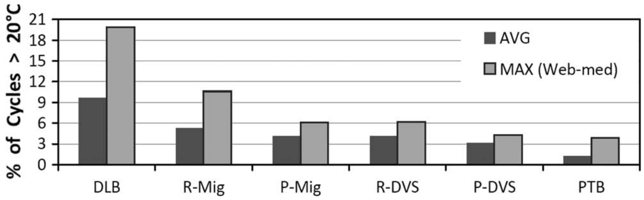

Fig. 13. Temperature cycles—with DPM (*simulator*).

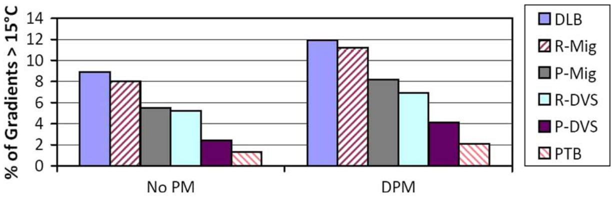

Fig. 14. Spatial gradients (*simulator*).

Our technique, PTB, achieves very similar thermal results to P-DVS while it has much better performance. DPM reduces the thermal hot spots to some extent, as it reduces the temperature when the system has idle time. Performing proactive temperature management results in the best thermal profile among the techniques when there is DPM, i.e., 83% reduction in hot spots in comparison to DLB.

We also looked at how the energy savings obtained with DPM change depending on the policy. Among the workload allocation/migration policies, DLB has the highest savings in energy. R-Mig, P-Mig, and PTB balance the workload more than DLB does to reduce thermal problems, whereas clustered workload achieves longer continuous idle time slots and helps DPM. DLB achieves 13.6% savings on average, while P-Mig and R-Mig reduce energy consumption by close to 12%. PTB performs dramatically better than the other migration-based policies in terms of reducing the thermal problems while still obtaining 8.9% savings when combined with DPM. DVS policies considerably increase the savings, e.g., 21.5% for P-DVS and 23.7% for R-DVS, when combined with DPM. However, recall that DVS significantly increases the execution time. Thus, while DVS reduces the energy consumption of cores, due to prolonged activity of memories and other components, the total energy consumption of the system may not benefit as much. Moreover, a significant portion of the total energy costs in current servers is due to cooling costs, which we did not consider in this computation.

Fig. 13 shows the average percentage of time we observed thermal cycles above 20 °C. We also plotted the workload with the maximum thermal cycling, i.e., Web-med, for comparison. We only consider the case with DPM for the thermal cycling results, as putting cores to the sleep state creates larger cycles. Our technique achieves very significant reduction in thermal cycles, i.e., to around 1% in the average case, as it continuously balances the workload among the cores according to their expected temperature. As reactive techniques take action after reaching temperature thresholds, they cannot avoid the temperature imbalance in time as much as our technique. P-DVS and PTB perform very similarly; however, it should be noted that the performance cost of PTB is less than DVS.

TABLE IV
TEMPERATURE RESULTS FOR THE COMBINED WORKLOADS (*simulator*)

	Hot Spots (%)	Cycles(%)	Gradients(%)
Web-med	2.6	4.5	4.4
Web&DB	4.6	2.9	5.7
Mplayer	0	0.1	0.9
(A) Web-med, Web&DB	3.7	3.7	5.0
(B) MPlayer, Web-med	1.3	2.2	2.7

Fig. 14 shows the average percentage of time we observed large spatial gradients above 15 °C. DPM creates larger gradients due to the low temperatures of the cores that go into the sleep state. PTB can almost eliminate large gradients by reducing their frequency to below 2% in average. P-DVS is the second best policy for reducing the on-die variations.

To show the effect of runtime adaptation on the accuracy of our technique, we ran traces of different workloads sequentially and computed the temperature statistics. In Table IV, we show the results for running the following combinations of workload with the PTB policy: (A) Web-med followed by Web&DB and (B) Mplayer followed by Web-med. We show the percentage of hot spots, cycles, and gradients for the individual workloads and also for the combined workloads for the case with DPM. We ran equal lengths of each benchmark in the combined workloads. We see that the percentage of hot spots and the variations of the combined workload are close to the average values of running the individual benchmarks. Thus, PTB can adapt to workload changes without negatively affecting the thermal profile.

We next discuss the results collected on the *real implementation*, where we implemented our technique in the Solaris task dispatcher running on an UltraSPARC T1 system. On the real implementation, we simulated DPM effects on temperature using HotSpot, as we did with the simulator, and assumed that the transition among active and sleep states has negligible

TABLE V
HOT SPOTS (*real implementation*)

	no PM				DPM			
Workload	DLB	R-Mig	P-Mig	PTB	DLB	R-Mig	P-Mig	PTB
Web-med	25.7	14.3	6.2	4.8	19.5	12.4	4.8	3.9
Database	8.4	3.5	1.8	1.3	4.6	3.1	1.5	1.2
Web&DB	32.4	15.7	8.1	6.0	27.4	14.7	9.1	5.8
Mplayer	4.9	1.5	0.7	0.9	1.9	2.0	1.5	1.2
(A)	17.2	9.1	4.9	4.1	23.7	14.2	7.9	5.1
(B)	15.6	8.5	4.5	3.7	10.7	8.0	3.7	3.6
AVG	17.4	8.8	4.4	3.5	14.6	9.1	4.8	3.5

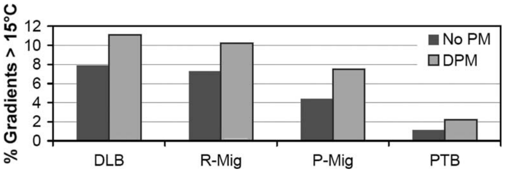

Fig. 15. Spatial gradients (*real implementation*).

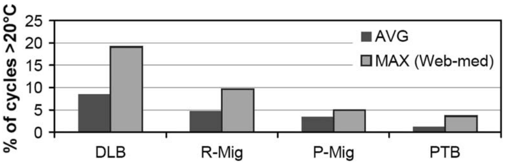

Fig. 16. Thermal cycles—with DPM (*real implementation*).

overhead. As the system does not have DVS capabilities, we simulated the thermal behavior for the default policy (DLB), reactive and proactive migrations, and our policy (PTB), running the benchmark set described previously.

In Table V, we show the distribution of hot spots, comparing various benchmarks. The combination workloads (A) and (B) are described in Table IV. We observe that PTB can reduce the hot spots by 60% in average in comparison to reactive migration and by 20% to 30% with respect to proactive migration. Workloads with low utilization, such as Mplayer, do not have a significant percentage of high temperatures. However, for hotter benchmarks, PTB achieves a dramatic reduction in the occurrence of hot spots.

Figs. 15 and 16 show the average frequency of spatial gradients and thermal cycles on our real system implementation. These results agree with the simulation results that PTB reduces the thermal variations more effectively in comparison to other proactive and reactive techniques.

As the *real implementation* on UltraSPARC T1 runs multithreaded workloads, we did not use an IPC-based performance metric. Evaluating the performance of multithreaded workloads using IPC is prone to inaccuracy [2]. This inaccuracy is due to the assumption that instructions per program remain constant across all executions, whereas the instruction path of multithreaded workloads running on multiple processors can vary substantially. Thus, to evaluate the performance of the various techniques we implemented, we used the "Load Average" metric. Load average is the sum of run queue length and number of jobs currently running. Therefore, if this number is low (i.e., typically below three or five, depending on the system), the response time of the system is expected to be fast. As load average grows, performance degrades.

Fig. 17 shows the performance values for the policies, normalized relative to the default policy (i.e., DLB). PTB is able to achieve better thermal profiles than other policies with less performance cost. This is because PTB first attempts to migrate the threads waiting in the dispatch queue, as opposed to stalling and migrating actively running threads. For example, for the workload Web−med, in the default case, the number of migrations of *active threads* was counted as 0.004 per 1000 instructions. Reactive migration (R-Mig) increases this number to 0.009/1000 instructions. Proactive migration causes fewer number of migrations than R-Mig (0.008/1000 instructions), as the temperature becomes more stable and the frequency of thermal emergencies decreases. PTB reduces this number further to 0.0046, which is only slightly higher than the default case. Note that the number of migrations of threads that are waiting in the queue is higher with PTB; however, the performance cost of such migrations is much lower.

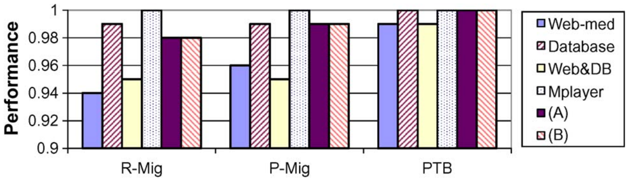

Fig. 17. Normalized performance (*real implementation*).

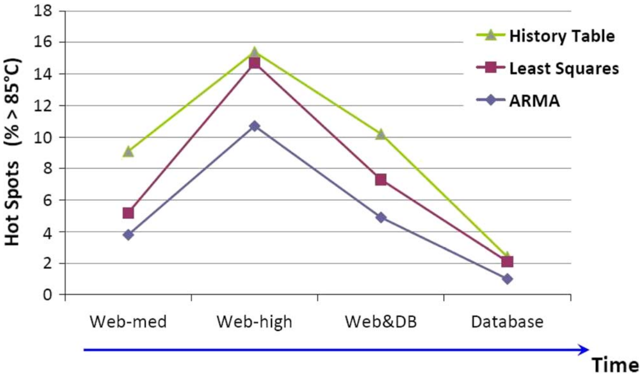

Fig. 18. Proactive balancing results for various predictors.

Lastly, to show the effect of prediction accuracy on thermal behavior, we implemented the PTB using least squares prediction and history predictor and compared the results against performing PTB with ARMA. Fig. 18 shows the percentage of hot spots observed with all the predictors. For this experiment, we ran the following benchmarks sequentially in the given order: Web-medium, Web-high, Web&Database, and Database. Each benchmark was run for an equal amount of time. PTB with ARMA achieves a better thermal profile than PTB with other predictors. This advantage is mainly a result of the longer adaptation period of the history and least squares predictors when the workload changes. SPRT detects the change immediately and computes a new ARMA model, whereas the other predictors go through a training period before starting accurate predictions.

B. Architecture-Level Simulator

To study the effect of reactive/proactive thermal management strategies in larger MPSoCs with higher performance cores, we have used an architecture-level simulator in addition to the results we collected on UltraSPARC T1. Following the trend of

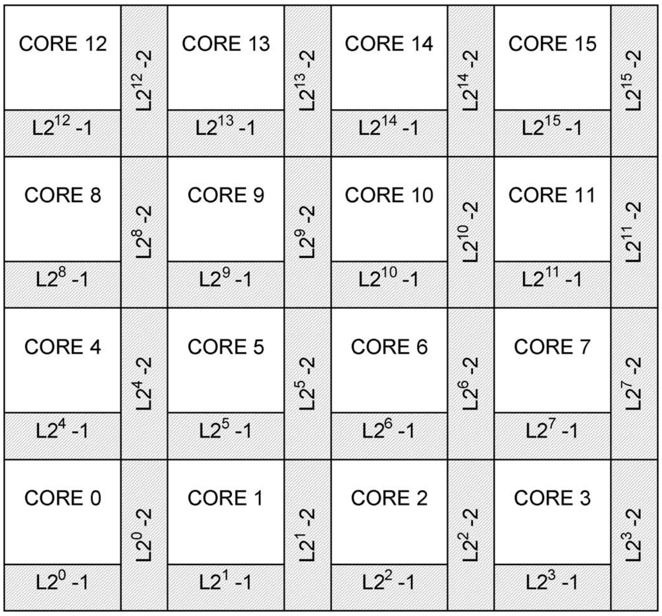

Fig. 19. Floorplan of the 16-core CPU [10].

integrating an increasing number of cores on a single die, e.g., Sun's 16-core Rock processor [33] and Intel's Larrabee with up to 32 cores [26], the CPU we model is a homogeneous 16-core multiprocessor manufactured at 65 nm. The floorplan for this CPU is shown in Fig. 19.

The simulation flow in this part consists of capturing the application phases using SimPoint [27] and computing the average power consumption for each phase. Then, with a finite number of simulation samples for each phase (using the M5 simulator [5] integrated with Wattch [6]), we reconstruct the power and execution properties of complete program execution. We do this for all voltage and frequency settings available, so that we can reconstruct the complete program if there are dynamic voltage/frequency changes. To model the power dissipation of L2 caches, we used CACTI [32]. This phase-analysis-based setup ensures that we can accurately simulate longer time frames than the typical architecture-level simulations.

We capture these program traces in a database which is queried by the scheduler at distinct intervals to determine the average IPC, power value, and the current instruction count. The power data collected by the scheduler are fed into HotSpot [28], which gives the thermal results for the modeled architecture. More details on the simulator are provided in [10].

We assumed that each core has three voltage and frequency settings for DVS: 1.2 V at 2.0 GHz, 1.187 V at 1.9 GHz, and 1.06 V at 1.7 GHz. Each core has a 64-kB two-way DCache and a 64-kB two-way ICache (each has an access time of two cycles). Each L2 cache (two banks; $L2^i - 1$ and $L2^i - 2$) is 2 MB, eight-way associative, and has an access time of 20 cycles. Memory latency is 200 cycles. Each core is single-threaded, has a four-width out-of-order issue, four integer arithmetic logic units (ALUs), two integer multiplication units, two FP ALUs, and two FP multiplier/dividers. The core architecture mimics an Alpha processor scaled to 65 nm. To model the penalty for thread migration, we measured the cold start effects for each benchmark (204 μs on average, reaching up to 1 ms). The delay for changing V/f level was set at 20 μs, based on the values reported in previous work [16].

TABLE VI
WORKLOAD CHARACTERISTICS FOR THE ARCHITECTURAL SIMULATOR

Workload	Benchmarks
1) 16-CPU	mesa*3, bzip2_program*3, crafty*2, eon_rushmeier*3, vortex1*2, sixtrack*3
2) 16-MIX	mcf*2, mesa, art110, sixtrack*2, equake, bzip2_program, eon_rushmeier*2, swim, applu, twolf, crafty, apsi, lucas
3) 14-CPU	mesa*2, bzip2_program*3, crafty*2, eon_rushmeier*2, vortex1*2, sixtrack*3
4) 14-MIX	mcf*2, mesa, art110, sixtrack*2, equake, eon_rushmeier*2 swim, twolf, crafty, apsi, lucas

We used benchmarks with different intensities of CPU and memory instructions to create representative traces for a wide range of real-world applications. We designed the following multicore workloads using the SPEC 2000 benchmark suite: 1) 16 CPU-bound threads; 2) 16 mixed threads (containing highly CPU-bound, highly memory-bound, and medium CPU-bound threads); 3) 14 CPU-bound threads; and 4) 14 mixed threads. The specifics of each workload are provided in Table VI.

The leakage model we use in this simulator is the same as described in Section VII-A. To get the transient temperature response, we integrated HotSpot [28] with Wattch. We calculated the die characteristics based on the trends reported for 65-nm process technology.

Next, we provide results on how the policies affect the thermal behavior and performance of the 16-core architecture. Fig. 20 shows the frequency of hot spots for R-Mig, P-Mig, DVS, and PTB. Note that, for this part of the experiments, we use the single-threaded version of the policies. Unlike the multithreaded simulations, in Fig. 20, we see that DVS can reduce the frequency of hot spots more effectively. However, this comes at a performance cost, which will be investigated later. On our 16-core architecture, we have not observed a significant amount of large temperature variations. The reason is that our applications highly utilized the system, unlike the multithreaded benchmarks with much more variant execution profile.

Next, we compare the performances of the temperature management techniques on the 16-core architecture using the fair speedup (FS) metric [7]. FS is computed by finding the harmonic mean of each thread's "speedup" over a baseline policy of running the thread at the highest frequency and voltage. Fig. 21 shows the performance for each workload and policy, as well as the average case for the 16-core architecture. PTB increases the performance by over 3% in comparison to P-DVS and by over 5% in comparison to R-DVS. PTB achieves the same performance as P-Mig while reducing the hot spots. Note that, on a single-threaded system, the performance benefit of PTB over P-Mig diminishes, as PTB is a policy that is specifically designed for optimizing multithreaded system performance.

On the 16-core system, we also ran simulations where the ARMA predictor was used for predicting IPC (as described in Section V). In this case, the proactive balancing policy was utilizing the IPC predictions (referred to as PTB_IPC). In other words, the prediction of high IPC is considered equivalent to a forecast of high power consumption. Therefore, the high-IPC threads are allocated to cooler locations. In the 16-core (4 × 4)

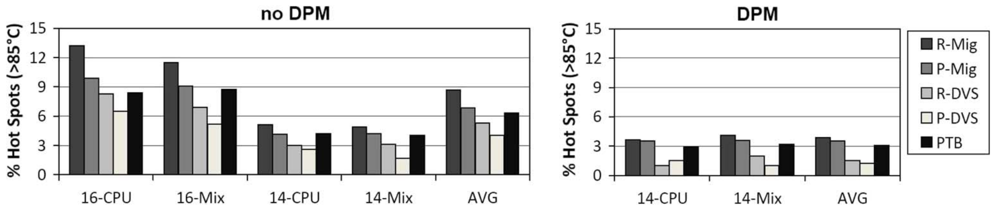

Fig. 20. Thermal hot spots (16-core system).

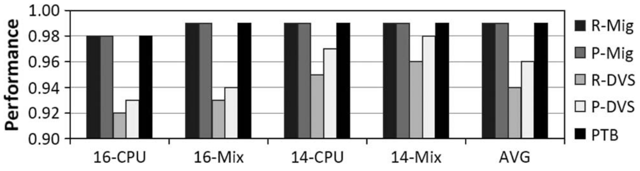

Fig. 21. Performance of policies on the 16-core architecture.

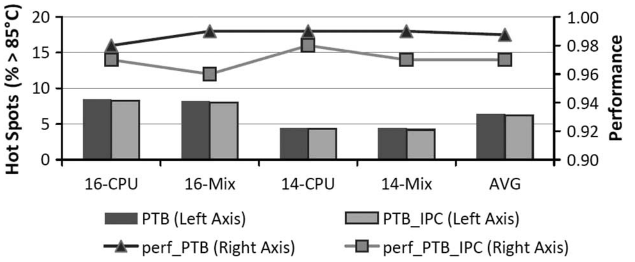

Fig. 22. Thermal results for ARMA IPC predictor.

MPSoC, the corner cores are typically cooler than other cores on the sides, and the cores at the center of the die are expected to be the hottest. How to order the cores in terms of their susceptibility to hot spots during scheduling has been discussed in prior work [10]. Note that, once the job with the highest IPC is allocated on one of the corner cores, the second highest IPC job will be allocated on the opposite corner (across the diagonal) to minimize the possibility of hot spots—this way, the policy avoids clustering the high power applications on neighboring cores. Thus, as PTB_IPC separates the high-IPC jobs from each other and places the lowest power jobs in the central region of the die, it is effective in reducing the frequency of hot spots.

Fig. 22 shows the thermal behavior achieved by PTB (temperature based) and PTB_IPC. The two techniques result in very similar percentages of hot spots, whereas PTB_IPC has higher performance overhead due to more frequent migrations. PTB_IPC reacts to changes in IPC, which are not always reflected to the temperature profile due to the thermal time constants. The results show that, for a single-threaded system without temperature sensors, IPC is a reasonable metric to guide thermal management. Note that, for other systems or workloads, PTB_IPC may result in higher percentage of hot spots as it does not consider the thermal interactions of neighboring units or the recent thermal history.

We observe that, in single-threaded MPSoCs, DVS has better results than job allocation policies (migration or balancing) in terms of reducing the hot spots. However, considering that the performance cost of DVS is higher, it would be beneficial to design hybrid strategies combining DVS and job scheduling to achieve a more desirable temperature–performance tradeoff. It should also be noted that DVS requires hardware support for the dynamic management of voltage, whereas the PTB we propose can be performed by only modifying the OS dispatching policy.

Previously in Section VII-A, we have seen that PTB accomplishes the reduction of the frequency of harmful temperature events as much as DVS while resulting in only a slight decrease in performance with respect to the DLB scheme. In multithreaded MPSoCs, proactive management brings significantly more benefits in reducing hot spots and temperature variations in comparison to applying the equivalent policies in single-threaded systems. This is due to the fact that multithreaded environment provides more opportunities for applying temperature-aware job allocation techniques without hurting performance.

VIII. Conclusion

In this paper, we have presented a proactive temperature management approach for MPSoCs, which can adapt to changes in system dynamics at runtime. We utilize ARMA modeling to accurately predict future temperature on each core based solely on the previous measurements. We continuously monitor how well the ARMA model fits the current temperature using SPRT and update the model if necessary. SPRT guarantees one to achieve the fastest detection of changes in thermal dynamics. Our PTB method for the dynamic allocation of threads reduces the thermal hot spots and temperature gradients significantly at very low performance impact. The proposed method does not require offline analysis or workload profiling and achieves more accurate predictions under dynamically variant workload in comparison to methods that rely on offline analysis or longer training periods.

We have provided a detailed comparison of our ARMA/SPRT-based approach to other prediction methods (e.g., exponential moving average, history predictor, and least squares predictor) and have also presented a thorough experimental evaluation of both reactive and proactive thermal management approaches on single-threaded and multithreaded MPSoCs. In our UltraSPARC T1 experiments, we have observed that our technique achieves 60% reduction in hot spot occurrences, 80% reduction in spatial gradients, and 75% reduction in thermal cycles in average, in comparison to reactive thermal management.

Acknowledgment

The authors would like to thank D. Atienza and the anonymous reviewers for their valuable feedback.

References

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

[2] A. R. Alameldeen and D. A. Wood, “IPC considered harmful for multiprocessor workloads,” *IEEE Micro*, vol. 26, no. 4, pp. 8–17, Jul./Aug. 2006.

[3] D. Atienza, G. D. Micheli, L. Benini, J. L. Ayala, P. G. D. Valle, M. DeBole, and V. Narayanan, "Reliability-aware design for nanometer-scale devices," in *Proc. ASPDAC*, 2008, pp. 549–554.

[4] D. Atienza, P. D. Valle, G. Paci, F. Poletti, L. Benini, G. D. Micheli, and J. M. Mendias, "A fast HW/SW FPGA-based thermal emulation framework for multi-processor system-on-chip," in *Proc. DAC*, 2006, pp. 618–623.

[5] N. L. Binkert, E. G. Hallnor, and S. K. Reinhardt, "Network-oriented full-system simulation using M5," in *Proc. Workshop CAECW*, 2003, pp. 36–43.

[6] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proc. ISCA*, 2000, pp. 83–94.

[7] J. Chang and G. S. Sohi, "Cooperative cache partitioning for chip multiprocessors," in *Proc. ICS*, 2007, pp. 242–252.

[8] A. K. Coskun, T. Rosing, and K. Gross, "Proactive temperature balancing for low cost thermal management in MPSoCs," in *Proc. ICCAD*, 2008, pp. 250–257.

[9] A. K. Coskun, T. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *Proc. DATE*, 2007, pp. 1659–1664.

[10] A. K. Coskun, R. Strong, D. Tullsen, and T. S. Rosing, "Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors," in *Proc. SIGMETRICS/Performance*, 2009, pp. 169–180.

[11] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *Proc. ISCA*, 2006, pp. 78–88.

[12] M. Gomaa, M. D. Powell, and T. N. Vijaykumar, "Heat-and-run: Leveraging SMT and CMP to manage power density through the operating system," in *Proc. ASPLOS*, 2004, pp. 260–270.

[13] K. Gross, K. Whisnant, and A. Urmanov, "Electronic prognostics through continuous system telemetry," in *Proc. MFPT*, Apr. 2006, pp. 53–62.

[14] K. C. Gross and K. E. Humenik, "Sequential probability ratio test for nuclear plant component surveillance," *Nucl. Technol.*, vol. 93, no. 2, pp. 131–137, Feb. 1991.

[15] W.-L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Thermal-aware task allocation and scheduling for embedded systems," in *Proc. DATE*, 2005, pp. 898–899.

[16] C. Isci, A. Buyuktosunoglu, C. CHer, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proc. MICRO*, 2006, pp. 347–358.

[17] C. Isci, G. Contreras, and M. Martonosi, "Live, runtime phase monitoring and prediction on real systems with application to dynamic power management," in *Proc. 39th MICRO*, 2006, pp. 359–370.

[18] "Failure mechanisms and models for semiconductor devices," *JEDEC Publication JEP122C*. [Online]. Available: http://www.jedec.org

[19] A. Karlin, M. Manesse, L. McGeoch, and S. Owicki"Competitive randomized algorithms for nonuniform problems," *Algorithmica*, vol. 11, no. 6, pp. 542–571, Jun. 1994.

[20] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha, "HybDTM: A coordinated hardware–software approach for dynamic thermal management," in *Proc. DAC*, 2006, pp. 548–553.

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

[22] *System Identification: Theory for the User*, 2nd ed. L. Ljung, Ed. Englewood Cliffs, NJ: Prentice–Hall, 1999.

[23] R. McDougall, J. Mauro, and B. Gregg, *Solaris Performance and Tools*. Upper Saddle River, NJ: Prentice–Hall, 2006.

[24] T. S. Rosing, K. Mihic, and G. D. Micheli, "Power and reliability management of SoCs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 4, pp. 391–403, Apr. 2007.

[25] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano, "Communication-aware allocation and scheduling framework for stream-oriented multi-processor system-on-chip," in *Proc. DATE*, 2006, pp. 3–8.

[26] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, "Larrabee: A many-core x86 architecture for visual computing," in *Proc. ACM SIGGRAPH*, 2008, pp. 1–15.

[27] T. Sherwood, G. H. E. Perelman, and B. Calder, "Automatically characterizing large scale program behavior," in *Proc. ASPLOS*, 2002, pp. 45–57.

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

[29] *SLAMD Distributed Load Engine*. [Online]. Available: www.slamd.com

[30] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," in *Proc. ISCA*, 2004, pp. 276–287.

[31] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif, "Full-chip leakage estimation considering power supply and temperature variations," in *Proc. ISLPED*, 2003, pp. 78–83.

[32] D. Tarjan, S. Thoziyoor, and N. P. Jouppi, "CACTI 4.0," HP Lab., Palo Alto, CA, Tech. Rep. HPL-2006-86, 2006.

[33] M. Tremblay and S. Chaudhry, "A third-generation 65 nm 16-core 32-thread plus 32-scout-thread CMT SPARC processor," in *Proc. ISSCC*, 2008, pp. 82–83.

[34] A. Wald and J. Wolfowitz, "Optimum character of the sequential probability ratio test," *Ann. Math. Stat.*, vol. 19, no. 3, pp. 326–339, 1948.

[35] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in *Proc. DAC*, Jun. 2008, pp. 734–739.

[36] Y. Zhang, X. S. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *Proc. DAC*, 2002, pp. 183–188.

Ayşe Kıvılcım Coşkun (S'06) received the B.S. degree in microelectronics engineering, with a minor degree in physics, from Sabanci University, Istanbul, Turkey, in 2003 and the M.S. degree in computer engineering from the Department of Computer Science and Engineering, University of California San Diego, La Jolla, where she is currently working toward the Ph.D. degree.

She was a Student Researcher with Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, for two summers—in 2005 and 2008. Since June 2006, she has been an Intern with Sun Microsystems. Her research interests are temperature and energy management in multiprocessor systems, 3-D stack architectures, reliable computer architectures, and embedded systems.

Ms. Coşkun has been a Reviewer for many prestigious IEEE and Association for Computing Machinery conferences and journals and has served in the review panel of the National Science Foundation.

Tajana Šimunić Rosing (M'90) received the M.S. degree in electrical engineering from the University of Arizona, Tucson, and the M.S. degree in engineering management and the Ph.D. degree from Stanford University, Stanford, CA, in 2001. Her Ph.D. topic was dynamic management of power consumption.

She was a Senior Design Engineer with Altera Corporation. She was a Full-Time Researcher with HP Labs while working part-time at Stanford University. She is currently an Assistant Professor with the Department of Computer Science and Engineering, University of California San Diego, La Jolla. Her research interests are energy efficient computing, embedded systems, and wireless systems.

Dr. Rosing has served at a number of technical paper committees and has been an Associate Editor of IEEE Transactions on Circuits and Systems. She is currently an Associate Editor of IEEE Transactions on Mobile Computing.

Kenny C. Gross (M'98) received the Ph.D. degree in nuclear engineering from the University of Cincinnati, Cincinnati, OH, in 1977.

He is a Distinguished Engineer with Sun Microsystems, San Diego, CA, leading the System Dynamics Characterization and Control team in Sun's Physical Sciences Research Center. He has 216 U.S. patents—issued and pending—and 169 scientific publications. He specializes in advanced pattern recognition, continuous system telemetry, and dynamical system characterization for improving the reliability, availability, and energy efficiency for computing systems.

Dr. Gross was the recipient of the 1998 R&D 100 Award for one of the top 100 technological innovations of that year, for an advanced statistical pattern recognition technique (called multivariate state estimation technique) that was originally developed for nuclear and aerospace applications and is now being used for a variety of applications to improve quality, availability, and energy efficiency for enterprise computer servers.