Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs

Ayse Kivilcim Coskun[†] Tajana Simunic Rosing[†] Kenny C. Gross[‡]
[†]University of California, San Diego [‡]Sun Microsystems, San Diego

Abstract

Conventional thermal management techniques are reactive in nature; that is, they take action after temperature reaches a predetermined threshold value. Such approaches do not always minimize and balance the temperature on the chip, and furthermore, control temperature at a noticeable performance cost. In this work, we investigate how to use predictors for forecasting future temperature and workload dynamics, and propose proactive thermal management techniques for multiprocessor system-on-chips (MPSoCs). The predictors we study include autoregressive moving average (ARMA) modeling and look-up table based predictors. We implement several reactive and predictive thermal management techniques, and provide extensive evaluations on an UltraSPARC T1 system as well as on an architecture-level simulator. By means of dynamic prediction, proactive methods can achieve significantly better thermal profiles in comparison to their reactive counterparts, and the performance cost associated with dynamic thermal management can be mostly avoided.

Utilizing Predictors for Efficient Thermal Management in Multiprocessor SoCs

I. Introduction

Chip power consumption is expected to increase with each new generation of computers while the geometries continue to shrink, causing higher power densities, thermal hot spots and large temperature variations on the die. Multiprocessor system-on-chips (MPSoCs) have enabled the designers to reduce the complexity and power consumption of CPU cores as MPSoCs' inherent thread level parallelism can achieve higher performance per Watt. However, as the performance demands grow, the number of cores integrated on a single die increase. In this work, we show that by performing thermal management and job allocation on an 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, and based on the predictions, they are able to act on thermal hot spots and temperature variations ahead of time.

High temperatures and large temperature variations cause a number of challenges for MPSoCs. Cooling costs continue to increase following the trend in power density. In addition to thermal hot spots, spatial thermal gradients on the die affect the cooling cost as large gradients decrease cooling efficiency. As leakage is exponentially related to temperature, increasing temperatures increase leakage power. Temperature also has a negative impact on performance since the operating speed of devices decreases with high temperatures. Finally, thermal hot spots and large temporal and spatial temperature gradients adversely affect reliability. Failure mechanisms such as electromigration, stress migration, and dielectric breakdown, which cause permanent device failures, are exponentially dependent on temperature [15]. Large spatial temperature gradients accelerate the parametric reliability problems caused by negative bias temperature instability (NBTI) and hot carrier injection (HCI) [17]. Temporal temperature fluctuations, i.e. high magnitude and frequency of thermal cycles, cause package fatigue and plastic deformations [15].

Previously proposed thermal management techniques typically focus on maintaining the temperature below critical levels. Methods such as clock gating and temperature-triggered frequency scaling [29] prevent hot spots when they are triggered by a temperature threshold. Obviously, such techniques come at a performance cost. In multiprocessor domain, techniques such as thread migration and applying PID control for keeping the temperature stable [8] have been introduced to achieve a safe die temperature at a reduced performance impact. Still, such techniques are reactive in nature; that is, they take action after temperature reaches a pre-determined 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. Especially in systems with dynamic power management (DPM) that turn off cores, reliability degradation can be accelerated because of the cycles created by workload rate changes and power management decisions [25]. Even though some dynamic temperature aware MPSoC scheduling techniques (e.g. [7]) are able to reduce thermal variations as well as hot spots in comparison to conventional thermal or power management, such techniques are reactive as well, and can be significantly improved by the dynamic forecasting of system behavior.

In this work, we design and evaluate proactive thermal management methods for MPSoCs to prevent thermal problems at negligible performance cost. 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. We compare ARMA predictor against other predictors (such as exponential averaging, recursive least squares [39] and history predictor) in terms of their forecasting accuracy. To address systems without temperature sensors, we utilize predictors to estimate future workload dynamics such as core utilization and IPC, and demonstrate the differences with temperature prediction.

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 [37]. Early detection of variations enable us to update the ARMA model for the current thermal dynamics immediately, and avoid inaccuracy.

Through utilizing the forecast for temperature or workload dynamics, the *proactive temperature balancing* technique we propose allocates incoming jobs to cores to minimize and balance the temperature on the die. To illustrate the advantages of our technique, we design and evaluate several reactive and predictive thermal management strategies for MPSoCS. We have performed experiments both on an UltraSPARC T1 [20], and also on an architecture-level simulator. We use real life workloads as measured by the Continuous System Telemetry Harness (CSTH) [10]. Proactive thermal management reduces the frequency of hot spots and large gradients significantly in

comparison to reactive thermal management strategies, while minimizing the impact on performance.

To summarize, the contributions of this work are:

- We demonstrate how to use various predictors (ARMA, history predictor and exponential averaging) to forecast future temperature and workload dynamics, and provide an extensive comparison of their prediction accuracy, overhead and adaptation capabilities.
- We utilize sequential probability ratio test (SPRT) to monitor temperature dynamics at runtime and adapt the ARMA predictor if the workload dynamics vary. We show that SPRT provides the fastest detection of changes in time series data.
- We propose a new proactive job allocation policy, which balances the workload among cores in an MPSoC based on the predicted temperature. The amount of balancing is proportional to the spatial temperature difference between cores, and we bound the number of thread re-allocations to reduce the performance overhead. In comparison to the state-of-art allocation methods used in modern OSes, our technique's performance overhead is negligible.
- We evaluate our proactive thermal management policy on an UltraSPARC T1 system and also on an architecture level simulator.

We start with a discussion of the related work in Section II. In Section III-A we provide the details of the ARMA based prediction method, and the online adaptation framework. We compare the ARMA predictor with other prediction methods in Section IV. Section V demonstrates how we can utilize online forecasting for predicting workload dynamics for systems without a sufficient number of thermal sensors. In Section VI, we explain all the thermal management techniques we study in this work, including our proactive MPSoC thermal management technique. Section VII provides the experimental methodology and results, and we conclude in Section VIII.

II. Related Work

In this section, we briefly discuss the techniques for multicore scheduling and thermal management. We also briefly investigate previous work on energy management, as the energy consumption affects the temperature significantly. In multicore systems, optimizing power-aware scheduling with timing and performance constraints introduces high complexity, as multicore scheduling is an NP-complete problem. A power management strategy for mission critical systems containing heterogeneous devices is proposed in [21]. A static scheduling method optimizing concurrent communication and task scheduling for heterogeneous network-on-chips is proposed in [12]. Rong et al. utilize integer linear programming for finding the optimal voltage schedule and task ordering for a system with a single core and peripheral devices [24]. In [26], MPSoC scheduling problem is solved with the objectives of minimizing the data transfer on the bus and guaranteeing deadlines for the average case. Minimizing energy on MPSoCs using DVS has been formulated using a two-phase framework in [40]. In [23], load balancing is combined with low power scheduling to reduce temperature in VLIW processors.

As power-aware policies are not always sufficient to prevent temperature induced problems, thermal modeling and management methods have been proposed. HotSpot [29] is an automated thermal model, which calculates transient temperature response given the physical characteristics and power consumption of units on the die. A fast thermal emulation framework for FPGAs is introduced in [3], which reduces the simulation time considerably while maintaining accuracy. Static methods for thermal and reliability management are based on thermal characterization at design time. Including temperature as a constraint in the co-synthesis framework and in task allocation for platform-based system design is introduced in [13]. RAMP [32] provides a reliability model at architecture level for temperature related failures, and optimizes the architectural configuration and power/thermal management policies for reliable design. In [25], it is shown that aggressive power management can adversely affect reliability due to fast thermal cycles, and the authors propose an optimization method for MPSoCs that saves power while meeting reliability constraints. A HW-SW emulation framework for reliability analysis is proposed in [2], a reliability-aware register assignment policy is introduced as a case study.

Dynamic thermal management controls over-heating by keeping the temperature below a critical threshold. Computation migration and fetch toggling are examples of such techniques [29]. Heat-and-Run performs temperature-aware thread assignment and migration for multicore multithreaded systems [9]. Kumar et al. propose a hybrid method that coordinates clock gating and software thermal management techniques such as temperature-aware priority management [18]. The multicore thermal management method introduced in [8] combines distributed DVS with process migration. In [33], dynamic MPSoC temperature-aware scheduling methods, which take core temperatures into account while making decisions, are proposed. The temperature-aware task scheduling method proposed in [7] achieves better thermal profiles than conventional thermal management techniques without introducing a noticeable impact on performance.

Our goal in this work is to 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 temperature reaches a certain level, our technique estimates the hot spots and temperature variations in advance, and modifies the job allocation to minimize temperature's adverse effects. For systems without temperature measurement hardware, we show how to predict workload dynamics and how to utilize proactive management for such cases. We also provide a comprehensive comparison of various reactive and proactive thermal management techniques in not only simulation but also on a real system implementation. In comparison to other prediction methods using history tables (as in [14]), our ARMA-based predictor can achieve much faster response to changing dynamics and also significantly increases the accuracy of predictions. The automated method we introduce for adaptation and predictor model computation provides a quicker and more robust prediction framework with respect to performing a recursive least squares fit as proposed in [39].

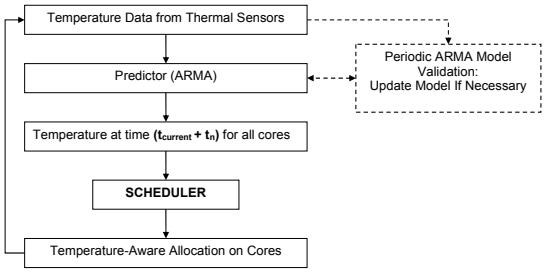

Fig. 1. Flow Chart of the Proposed Technique

III. Prediction with Autoregressive Moving Average (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. Proactive management adjusts the workload distribution on an MPSoC using the predictions. This way, we prevent thermal problems before they occur, as opposed to reacting to hot spots and variations after they appear on the system. Hence, we can reduce the temperature induced problems much more effectively at negligible performance cost.

Figure 1 provides an overview of our technique. Based on the temperature observed through thermal sensors, we predict temperature t_n steps into the future using an autoregressive moving average (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.

Autoregressive moving average (ARMA) models are mathematical models of autocorrelation in a time series. They are widely used in many fields for understanding the physical system or forecasting the behavior of a time series from past values alone. In our work, we use ARMA models to predict future temperature of cores using the observed temperature values in the past. ARMA model assumes the modeled process is a stationary stochastic process, and that there is 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 the underlying data for the ARMA model is stationary. We adapt the model when the ARMA model no longer fits the workload. Thus, the stationary assumption does not introduce inaccuracy.

An ARMA model is described by Equation 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 coefficient, c_i is the moving average coefficient and e_t is called the noise, error or the 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 autoregressive (AR) and the moving average (MA) parts of the model, respectively. For example, a first order ARMA model is written as: $y_t + (a_1\ y_{t-1}) = e_t + (c_1\ e_{t-1})$.

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

ARMA modeling has the following steps: 1)*Identification*, which consists of specifying the order of the model; 2) *Estimation*, which is computing the coefficients of the model (generally performed by software with little user interaction); 3) *Checking the Model*, where it is ensured that the residuals of the model are random, and that the estimated parameters are statistically significant.

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, for fitting the data with the current order of ARMA model, coefficients are computed using a least-squares fit. Other methods could also be utilized for coefficient estimation.

We use *Final Prediction Error (FPE)* 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 Equation 2, 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.

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

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, 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 pre-determined confidence intervals.

As an example, we have applied the ARMA prediction methodology to a sample temperature trace. The trace is obtained through HotSpot [29] 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 in Figure 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

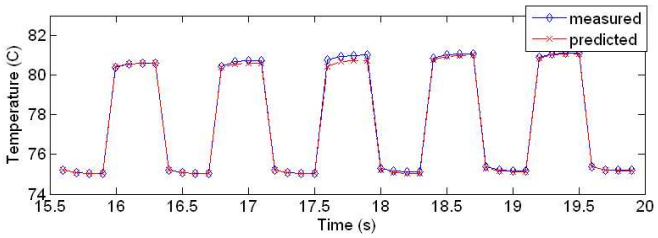

Fig. 2. Temperature Prediction

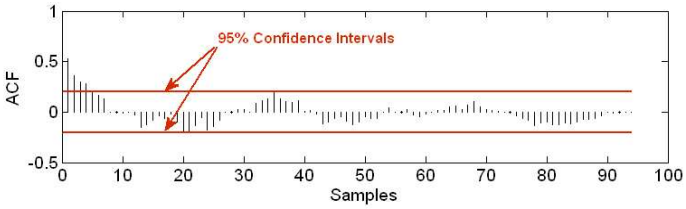

Fig. 3. Autocorrelation Function of the Residuals

data, we formed ARMA models with $FPE << 1$. It should be noted that much shorter training sets (i.e., 20-50 samples) are sufficient for forming an ARMA model with the desired fit for most real-life workload traces we experiment with.

We saved the last 50 samples of the data to test our prediction method. We used the ARMA model to predict 5 steps (i.e. 500 ms) into the future. The prediction results are demonstrated in Figure 2. The prediction matches the observed values closely. For temperature curves with less temporal variation, designing an accurate ARMA predictor is even easier.

Figure 3 shows the ACF of the residuals for the model in Figure 2. The ACF of residuals fluctuate around zero, showing that the residuals of the model are random. The horizontal lines in the figure show the 95% confidence intervals. In our automated methodology, we observe 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 no moving-average component can be computed in less than 150ms, and an ARMA(p,q) model up to 5^{th} order can be computed in less than 300ms. The computation and the validation of the model together takes between 250 and 500ms. Note that implementing the ARMA process in C/C++ and optimizing the source code would significantly reduce the overhead.

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 the current workload deviates from the initial assumptions used for forming the ARMA model, we update the model on the fly.

We use Sequential Probability Ratio Test (SPRT) to detect changes over time in statistical characteristics of the residual signals. SPRT test on the residuals provides the earliest possible indication of anomalies [37], where as the anomaly in this case 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 user specified false-alarm 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 observed residuals appears to be distributed about mean $+M$ or $-M$ instead of around 0, 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, let us define the residuals by Equation 3, where $T_i^{'}(t)$ is the prediction and $T_i(t)$ is the actual measurement.

$$R(t) = T_i(t) - T_i^{'}(t) \tag{3}$$

SPRT enables us to decide between the following two hypothesis:

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 drawn from a pdf with mean 0 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 false alarm and missed alarm probabilities. Small values such as 0.01 or 0.001 are used for α and β.

SPRT is applied to detect the drift (i.e. anomaly) in residuals by computing the log likelihood ratio in Equation 4.

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

where $p(./H_2)$ is the joint density function assuming no fault, and $p(./H_1)$ is the joint density function assuming fault, and N is the number of observations. If $LR_N \geq B$ we accept

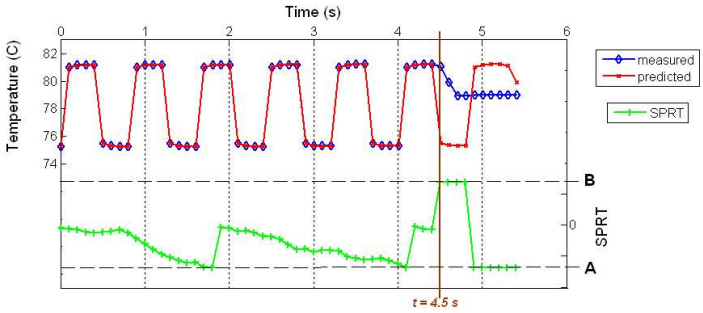

Fig. 4. Online Detection of Variations in Thermal Characteristics

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 hypothesis 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 Equation 5.

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

Following the derivation provided in [11], the value of SPRT can be represented as in Equation 6, and the bounds demonstrated in Equation 5 to make decisions. 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.

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

Note that M and σ^2 values are computed at the beginning, and then fixed until the ARMA model is updated. So at runtime, during each sampling interval, the SPRT equation effectively performs one addition and one multiplication. Because of the simplicity of computation shown in Equation 6, the cost of computing SPRT after each observation is very low (negligible in our measurements). Moreover, as shown in [37], there is no other procedure that has the same error probabilities with shorter average sampling time than SPRT. Thus, we have picked SPRT as the online monitoring tool in this work due to both its guarantee for fast detection of changes and low computation overhead.

In Figure 4, we demonstrate a case where the temperature dynamics change, and the SPRT detects this change immediately (see $t = 4.5s$ 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.

We also compared SPRT detection to 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 (i.e. difference between actual measurements and predictions). 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

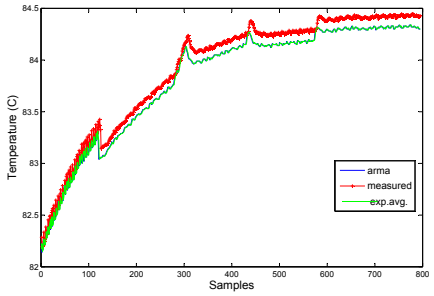

Fig. 5. Comparison of Predictors - Stable Temperature

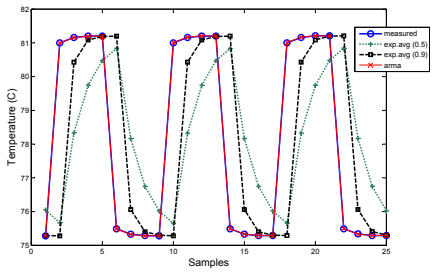

Fig. 6. Comparison of Predictors - Thermal Cycling

the standard deviation of prediction errors 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 Figure 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

In Figures 5 and 6, we compare ARMA prediction with exponential average prediction. 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 ($0 \leq \alpha \leq 1$). In these comparisons, we used $\alpha = 0.9$ and $\alpha = 0.5$.

When the change in temperature is slow and we have relatively stable temperature, exponential average predictor works well, providing almost the same values as the ARMA predictor in Figure 5. However, when there are rapid temperature changes, such as thermal cycling, exponential average predictor performs poorly, such as in Figure 6. In addition, even though exponential average predictors with $\alpha = 0.9$ and $\alpha = 0.5$ perform very similarly in the first example, there is 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

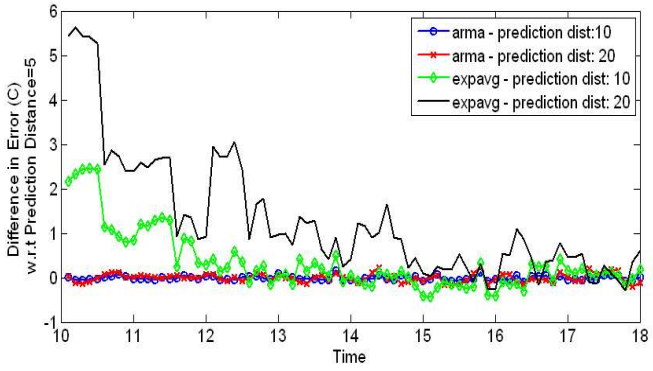

Fig. 7. Predicting Further Ahead with Exponential Averaging

forming the model with high accuracy. The overheads of evaluating the ARMA or the exponential average model at runtime are very similar, as both models only compute a polynomial equation for each sample.

Another disadvantage of the exponential predictor is that, when the goal is forecasting several time steps into the future, the prediction accuracy degrades significantly. In Figure 7, we demonstrate how the accuracy of ARMA compares to exponential averaging for forecasting 10 and 20 time steps ahead. For this experiment, we used a 200 sample temperature trace for a CPU bound SPEC 2000 suite workload. In this trace temperature was changing within a 1.5^oC range. The plot shows the difference of error (in ^oC) in comparison to that predictor's error when predicting 5 steps into the future. While ARMA predictor's accuracy is stable, the error margin of the exponential predictor decreases significantly when predicting ahead.

B. History Predictor

In Section III, we showed that it is possible to predict future temperature accurately based on the previous thermal measurements. Following this insight, we built a history predictor. A history predictor is similar to a global branch predictor, and it consists of a shift register that tracks the last few observed values. The length of the history is specified by shift register depth. At each sampling period, the register is updated with the last 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 based on previous experience. 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. An invalid entry is kept for each tag to track the ages of different HT tags for a least recently used (LRU) replacement policy when the HT is full. A -1 entry denotes the corresponding tag contents and prediction are not valid. This predictor is similar to the Global History Predictor used for predicting power phases in [14]. When the shift register does not hit the history, the predictor behaves like a last-value predictor, and assumes the future temperature value will be the same as the last observed temperature.

One issue with the history predictor is regarding the decimal places of the temperature. We have performed experiments

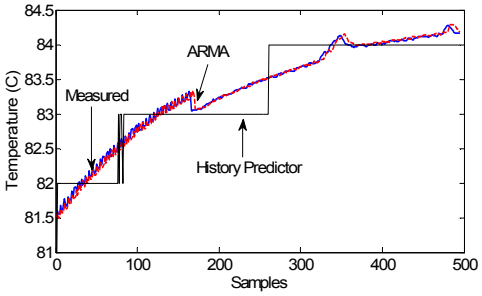

Fig. 9. Comparison of ARMA and History Predictor

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 significantly 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 nearest integer values, and only predict temperature in integers.

In Figures 8 a and b, we show the accuracy for various history table (HT) sizes and history lengths. In this experiment, we have used the same temperature trace we used for Figure 7, and predicted 5 steps ahead. In (a), we compare the standard deviation of error and mean error (in ^oC) for the prediction, where error is the difference of 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 (b), we demonstrate the correct prediction ratio (with respect to the integer temperature trace) and the hit rate for the history table. 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, last value predictor compensates well when the history predictor cannot predict. Another outcome to note is that, increasing the table size over 100 does not bring additional benefits, which motivates using a small-size table to achieve enough accuracy with lower hardware overhead.

Figure 9 compares the ARMA predictor and the history predictor (HP with HT size of 100 and history length of 5) for predicting 5 steps (i.e., 500ms). For 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 history table is large enough to maintain the entries associated with all of the applications.

C. Recursive Least Squares

Another temperature prediction method recently utilized in [39] is using recursive least squares. In Figure 10, we compare the prediction accuracy of the least square approach versus the ARMA predictor. We trained both predictors with 50 samples of the temperature data, and started the predictions

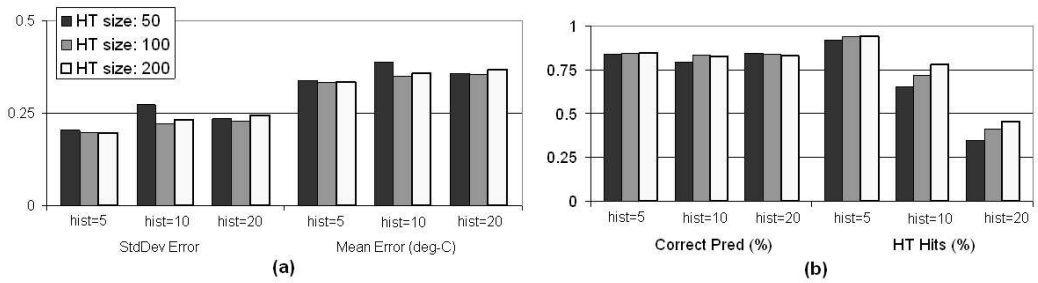

Fig. 8. Accuracy-Size Trade-Off for the History Predictor

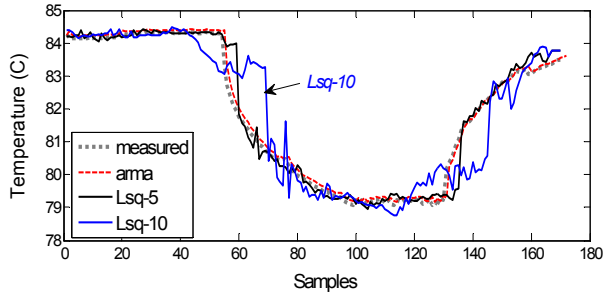

Fig. 10. Comparison of ARMA and Recursive Least Squares Predictor

afterwards. As we observe in the figure, while the least square predictor can predict with similar accuracy for the prediction distance of 5 (shown as *Lsq-5*), the accuracy drops rapidly when we increase the time window we are forecasting into the future (t_n) to 10 steps. This trend continues even more dramatically as the forecasting distance is increased.

Note that both of the above methods can predict the data in the history window they are trained with with any desired degree of accuracy. If one keeps adding enough terms, it is even possible to fit through every single observation in the history window. Typically we would not want to do that, however, because typically there is random measurement noise on the time series, and there is no value to learning the noise. Thus, the differentiating point of least squares and ARMA arises when we are forecasting further into the future. As we increase t_n, 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 onwards.

Another important advantage of ARMA in comparison to least squares is in overhead. Recursive least squares method continuously updates the coefficients of the model as new data arrives (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 automatically 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 prediction of two parameters: 1) Core utilization, 2) 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, especially in multi-threaded 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.

In Figures 11 and 12, we show 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 prediction was performed for the following 50 samples. One issue to note is 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 especially 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 before performing prediction. The smoothed-out utilization and prediction signals are demonstrated with the subscripts sm in Figure 11. We observe that the accuracy of prediction is significantly lower than temperature prediction. However, when the data is smoothed-out first, the predictor works more accurately.

Even though bzip is a highly IPC-variant benchmark, Figure 12 shows that IPC can be predicted quite accurately. 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 multi-threaded system, where the core is running a set of threads rather than a single application.

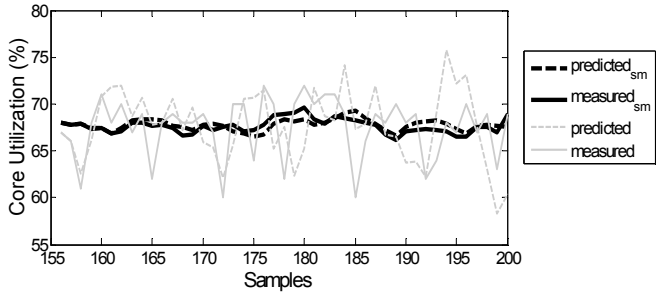

Fig. 11. Prediction of Core Utilization

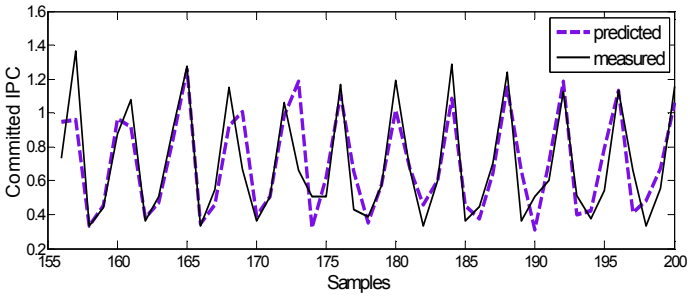

Fig. 12. Prediction of Committed IPC

VI. Proactive Temperature Management

We perform an extensive study of various thermal management techniques for MPSoCs, and propose a proactive temperature-aware job allocation technique. In this section, we discuss the details of all the techniques we implemented. We consider both single-threaded and multi-threaded systems while studying thermal management policies. In the system model for the multithreaded systems, each core is associated with a *dispatching queue*, which holds the threads allocated to that core. This is the typical abstraction used in modern multicore OS schedulers, which are based on multilevel queuing. The dispatcher allocates the incoming threads to queues based on the current policy. Many temperature management methods rely on dynamic temperature information acquired from the system, so we assume temperature readings are available for each core through thermal sensors. The management policies observe the system characteristics at regular intervals (i.e. ticks) to make decisions.

The default policy we evaluate (i.e. default policy in modern OSes, 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 significant imbalance among the queues, the threads are migrated to have more balanced utilization.

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; `Dynamic Power Management (DPM)` and `Dynamic Voltage-Frequency Scaling (DVS)`. For DPM, we utilize a fixed timeout policy, which puts a core to sleep state if it has been idle longer than the timeout period. We set the timeout as the breakeven time [16]. The DVS policy observes the core utilization over a given length of recent history, and reduces the frequency/voltage proportionally.

Reactive Thermal Management:

Several reactive thermal management techniques have been proposed in the literature (e.g. [9]). In this work we implement some of the most commonly used methods.

- `Reactive Thread Migration` migrates the workload from a core if the threshold temperature is exceeded to the coolest core available. In single threaded systems, this correspond 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` reduces the voltage/frequency (V/f) setting on a core if the threshold temperature is exceeded, similar to the frequency scaling approach in [29]. We assume three built-in voltage frequency 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.

Proactive Thermal Management:

The proactive methods we discuss next utilize the temperature prediction introduced in Section III-A. The motivation behind proactive management is to respond to thermal emergencies before they occur, and thus to minimize the adverse affects 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. Our work focuses on finding effective dispatching methods to reduce temperature induced problems without affecting performance.

- `Proactive Thread Migration` moves workload from cores that are projected to be hot in the near future to cool cores. `Proactive 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` follows the principle of locality (i.e. allocating the threads on the same core they ran before) during initial assignment as in the default policy. At every scheduler tick, if the temperature 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 before they occur, 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.

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 have not been migrated during the current tick. Note that in a multi-threaded environment, threads waiting in the queue are moved unless the threshold is already exceeded, so migration does not stall the running thread. This is in contrast to moving the actively running threads in thread migration policies discussed above. 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 Proactive Temperature Balancing for multi-threaded systems, the number of threads to migrate is proportional to the spatial temperature difference among the hot core and the cool core.

VII. Experimental Results

In this section, we evaluate the thermal management techniques we have discussed in the previous section. In the results, DLB is the default load balancing policy, R-Mig., P-Mig. and DVS are the proactive and reactive migration and voltage scaling, respectively, and PTB is the proactive temperature policy we propose.

We show two sets of experimental results. The first set is based on the UltraSPARC T1 processor [20]. 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 65nm.

In the experimental evaluation, the hot spot results demonstrate the percentage of "time spent above 85^oC", which is considered a high temperature for our system. Similar metrics have been used in previous work as well (e.g.[19]). The spatial gradient results summarize the percentage of time that gradients above 15^oC occur, as gradients of $15-20^oC$ 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 to 20^oC, failures happen 16 times more frequently [15]. So, we report the temporal fluctuations of magnitude above 20^oC. ΔT values we report are computed over a sliding window and averaged over all cores.

A. UltraSPARC T1 Implementation

The first set of experimental results are based on the UltraSPARC T1 [20]. 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.

In the results marked as *real implementation*, we implemented the policies in Solaris and ran the workload on the UltraSPARCT1 in real time. We simulated the power/thermal dynamics simultaneously on another machine running in parallel. The results marked as *simulator* are from our simulation infrastructure attached to the power/thermal model, where we use the real-life workload traces again, but implement the scheduling policies within the simulator on a replica of the multicore system model.

We leveraged the Continuous System Telemetry Harness (CSTH) [10] to gather detailed workload characteristics of real applications. We sampled the utilization percentage for each hardware thread at every second using `mpstat` [22]. We recorded half an hour long traces for each benchmark. To determine the active/idle time slots of cores more accurately, we recorded the length of user and kernel threads using `DTrace` [22].

We ran the following sets of benchmarks: 1) Web server, 2) Database, 3) Common Integer, 4) Multimedia. To generate web server workload, we ran SLAMD [30] with 20 and 40 threads per client to achieve medium and high utilization, respectively. For database applications, we tested MySQL using `sysbench` for a table with 1 million rows and 100 threads. We also ran compiler (`gcc`) and compression/decompression (`gzip`) benchmarks. For multimedia, we ran mplayer (integer) with a 640x272 video file. We summarize the details of our benchmarks in Table I. The utilization ratios are averaged over all cores throughout the execution. Using `cpustat`, we also recorded the cache misses and floating point (FP) instructions per 100K instructions.

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

Peak power consumption of SPARC is similar to the average power [20], so we assumed that the instantaneous power consumption is equal to the average power at each state (active, idle, sleep). The average power consumption for UltraSPARC T1 (including leakage) and area distribution of the units on the chip are provided in Table II, and the floorplan is available in [20].

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 [34]. We determined the coefficients in the model empirically to match the normalized leakage values in the paper. 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 Watts, which is estimated based on sleep power of similar cores. To compute the power consumption of the crossbar, we scaled power according to the number of active cores and the memory access statistics.

We used HotSpot version 4.0 [29] as the thermal modeling tool, and modified the floorplan and thermal package characteristics for UltraSPARC T1. We performed the thermal simulations with a sampling interval of 100 ms, which provided good precision. We initialized HotSpot with steady state temperature values.

First we provide the results obtained on our *simulator*. 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^oC for all the workloads, and the average results for the cases with no power management (No PM) and DPM. The performance results shown in the table are normalized with respect to the default policy's performance. We computed performance based on the average delay we observed in the thread completion times. Migration of workload upon reaching the threshold or applying temperature triggered DVS cannot eliminate all the hot spots, especially for workloads with medium to high utilization level. Performing migration or DVS proactively achieves significantly better results, while also reducing the performance cost. The performance overhead is less with the proactive approaches as they maintain the temperature at lower levels, requiring fewer overall number of migrations or shorter periods of DVS. Our technique, PTB, achieves very similar results to proactive 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, as demonstrated in the table.

Figure 13 shows the average percentage of time we observed thermal cycles above 20^oC. 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 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 well 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.

Figure 14 shows the average percentage of time large spatial gradients above 15^oC occurred while running the policies. DPM creates larger gradients due to the low temperatures on the cores that go into the sleep state. Proactive temperature balancing can almost eliminate large gradients by reducing their frequency to below 2% in average. Proactive DVS is the second best policy for reducing the on-die variations.

To show the effect of adaptation (i.e. when workload changes) on the accuracy of our technique, we ran traces of different workloads sequentially and computed the temperature statistics. As examples, 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, (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 variations of the combined workload are close to the average values of running the individual benchmarks. This shows us that PTB can adapt to workload changes without negatively affecting the thermal profile.

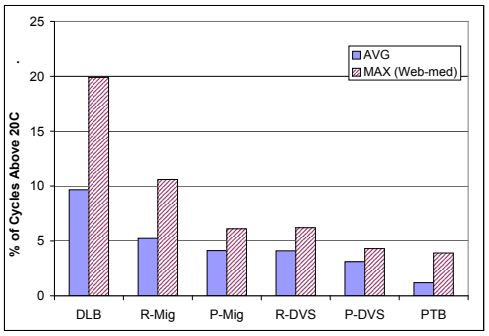

Fig. 13. Temperature Cycles - with DPM (*simulator*)

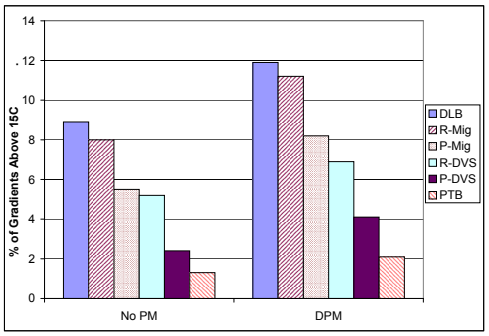

Fig. 14. Spatial Gradients (*simulator*)

TABLE IV
TEMPERATURE RESULTS FOR 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

We next discuss results collected on the *real implementation*, where we implemented our technique in the Solaris task dispatcher running on an UltraSPARC T1 system. Some of our policies utilize temperature readings from all cores, and UltraSPARC T1 does not contain an individual sensor for each core. To obtain per core temperature data, we installed HotSpot on another machine in the private network, to avoid introducing performance overhead on the system we are running the experiments on. Based on the utilization of each core and the average power values, HotSpot computes the temperature values and communicates them back to the system running the

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

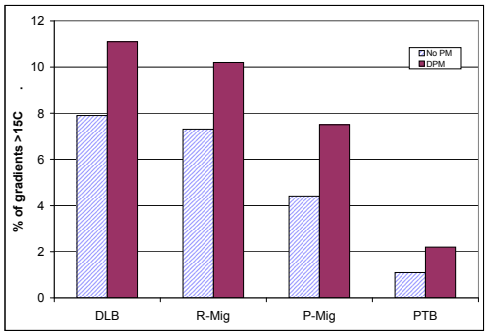

Fig. 15. Spatial Gradients (*Real Implementation*)

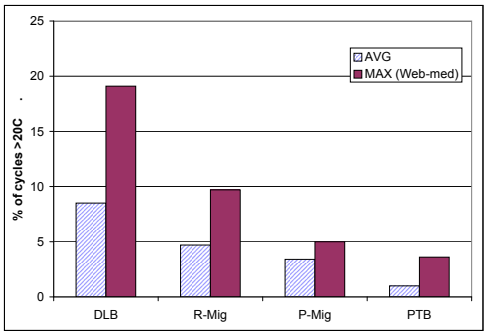

Fig. 16. Thermal Cycles - with DPM (*Real Implementation*)

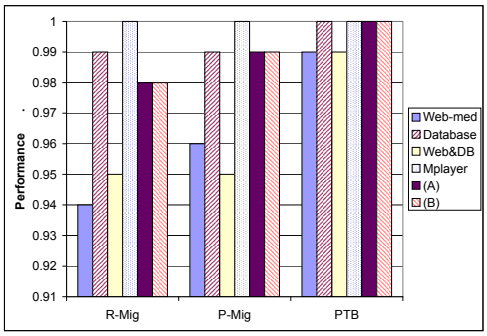

Fig. 17. Normalized Performance (*Real Implementation*)

thermal management techniques through a shared file between the two machines. We used an interval length of 1 second in these experiments.

We simulated DPM effects on temperature using HotSpot as we did with the simulator, and assumed the transition overhead among active and sleep states has negligible overhead. As the system does not have DVS capabilities, we simulated the thermal behavior for the default policy (DLB), reactive and proactive migration, and our policy (PTB), running the same benchmarks 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 above. We observe that PTB can reduce the hot spots by 60% in average in comparison to reactive migration, and 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 dramatic reduction in the occurrence of hot spots.

In Figures 15 and 16, we demonstrate the average frequency of spatial gradients and thermal cycles on our real system implementation. These results confirm the simulation results that, our proactive policy, PTB, reduces the spatial and temporal variations in temperature in comparison to other proactive and reactive thermal management techniques.

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 3 or 5, depending on the system), the response time of the system is expected to be fast. As load average grows, performance gets worse. Figure 17 demonstrates the performance values for the policies, normalized relative to the default policy (i.e. dynamic load balancing). Proactive temperature balancing is able to achieve better thermal profiles than other policies with less performance cost. This is due to the fact that PTB first attempts to migrate the threads waiting in the dispatch queue, as opposed to stalling and migrating actively running threads.

Lastly, to show the effect of prediction accuracy on thermal behavior, we implemented the proactive temperature balancing (PTB) technique using least squares prediction (LSQ) and history predictor (HISTORY), and compared the results against performing PTB with the ARMA predictor. Figure 18 compares the percentage of hot spots observed with all the predictors. For this experiment, we ran a sequential trace of several benchmarks in the following order: Web-medium, Web-high, Web&Database and Database. Each benchmark was run for an equal amount of time. ARMA predictor combined with PTB achieves a better thermal profile than using the other predictors. This advantage is mainly a result of the longer adaptation period the history predictor and least squares

TABLE V
HOT SPOTS (EM 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

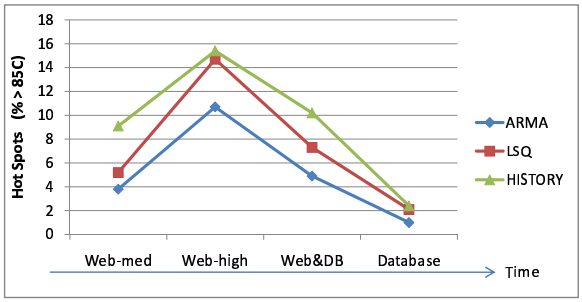

Fig. 18. Proactive Balancing Results for Various Predictors

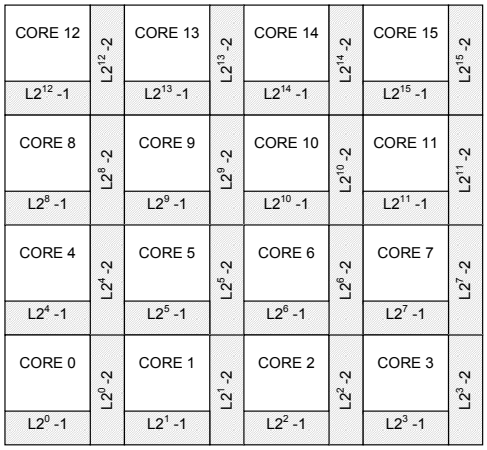

Fig. 19. Floorplan of the 16-core CPU

predictor spend when the workload changes. SPRT detects the change in workload immediately and initiates the computation of a new ARMA model, whereas the other predictors have to go through a training period to be able to provide accurate predictions.

B. Architecture-Level Simulator

To study the effect of reactive and proactive thermal management strategies in larger MPSoCs with higher performance cores, we have used an architecture-level simulator in our experimental work in addition to the results we collected on UltraSPARC T1. Following the industry trend of integrating an increasing number of cores on a single die, e.g. Sun's 16-core Rock processor [36] and Intel's Larrabee with up to 32 cores [27], the CPU we model is a homogeneous 16-core multiprocessor manufactured at 65nm. The floorplan for this CPU is provided in Figure 19. Each core has out-of-order issue, a private data cache, instruction cache, L2 cache, and memory channels.

The simulation flow in this part consists of capturing the application phases using SimPoint [28], and computing the average power consumption for each phase. Then, with a finite number of simulation samples for each phase (using the M5 simulator [4] integrated with Wattch [5]), 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 even in the face of an arbitrary number of voltage/frequency changes. To model power dissipation of L2 caches, we used CACTI [35]. This phase analysis based set-up is used to ensure that we can simulate longer time frames than typically architecture-level simulations with high accuracy.

We capture these program traces in a database which is queried by the scheduler at distinct intervals. Given a program start point, an interval length in cycles, and a frequency setting, the query tool returns the average IPC and power levels across the interval, and the point in execution the program reaches at the end of the interval. The power data collected by the scheduler is fed into HotSpot [29], which gives the thermal results for the modeled architecture.

We assumed each core possesses three voltage and frequency settings for dynamic voltage/frequency scaling (DVS), similar to our assumptions in the previous experimental setup: 1.200V at 2.0GHz, 1.187V at 1.900GHz, and 1.06V at 1.7GHz. Each core has a 64KB 2-way DCache and 64KB 2-way ICache. Size of each L2 cache is 2MB. Each core has single-threaded execution and 4-width out-of-order issue, 4 integer ALUs, 2 integer multiplication units, 2 FP ALUs and 2 FP Multiplier/Dividers. The core architecture mimics an Alpha processor scaled to 65nm.

We used applications with different characteristics of CPU and memory boundedness [38] in our simulations to create representative traces for a large 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, 4) 14 mixed threads. The specifics of each workload is 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, HotSpot [29] was integrated with Wattch. We calculated the die characteristics based on the trends reported for

TABLE VI
WORKLOAD CHARACTERISTICS FOR THE ARCHITECTURAL SIMULATOR

Workload name	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

65nm process technology.

Next, we provide results on how the policies affect the thermal behavior and performance of the 16-core architecture. Figure 20 demonstrates 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 multi-threaded simulations, in Figure 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. This is due to the fact that our applications we used highly utilized the system, unlike the multi-threaded benchmarks that have a much more variant execution profile.

Next, we compare the performance of the temperature management techniques on the 16-core architecture. A common performance metric on multicore platforms is a raw count of instructions per second, frequently measured in millions or billions (BIPS). However, this metric gives undeserved bias towards high-IPC threads as performance may be increased by running more CPU bound threads. To circumvent this difficulty, we used the Fair Speedup Metric (FS) from [6] and [31]. FS is computed by finding the harmonic mean of each thread "speed-up" over a baseline policy of running the thread at highest frequency and voltage.

Figure 21 provides the performance hit computed 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, as described earlier. It should also be noted 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.

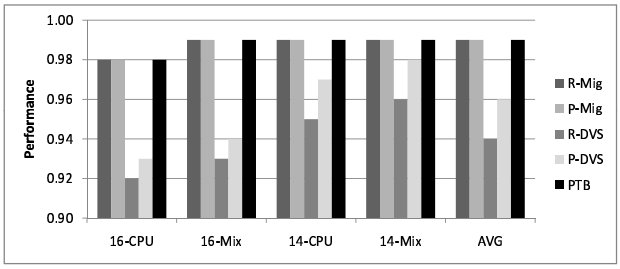

Fig. 21. Performance Cost of Policies on the 16-core architecture

On the 16-core system, we also ran simulations where the ARMA predictor was used for predicting IPC, and the proactive balancing (PTB) was ran based on the IPC predictions instead of temperature. In other words, prediction of high IPC is considered equivalent to a forecast of high temperature. Therefore, the high-IPC threads is moved to cooler cores. Again, we follow the same principle as in PTB with temperature prediction for bounding the number of migrations. Figure 22 shows the thermal behavior achieved by PTB_Thermal and PTB_IPC, which refer to PTB combined with temperature forecast and IPC forecast, respectively. PTB_IPC achieves a very slight (negligible) reduction in hot spots at the cost of higher performance overhead. The reason for the impact in performance is the increased number of migrations. In PTB_IPC, the policy reacts to changes in IPC which may not get 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.

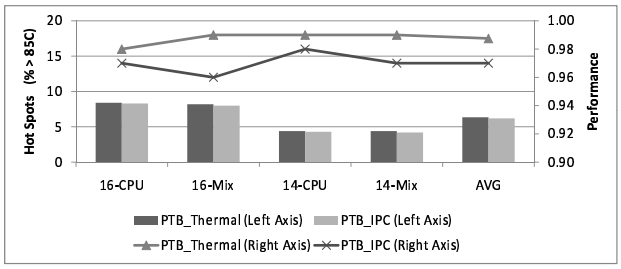

Fig. 22. Thermal Results for ARMA IPC Predictor

VIII. CONCLUSION

In this paper, we have presented a proactive temperature management approach for multiprocessor system-on-chips (MPSoCs), which can adapt to changes in system dynamics at runtime. We utilize autoregressive moving average (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 sequential probability ratio test (SPRT), and update the model if necessary. The advantage of SPRT is that it guarantees to achieve the fastest detection of changes in thermal dynamics. Our proactive temperature balancing method for dynamic allocation of threads is able to reduce the thermal hot spots and temperature gradients significantly at very low performance impact. The proposed method is a completely online approach, and does not require offline analysis or workload profiling.

Our paper provides a thorough experimental evaluation of reactive and proactive thermal management approaches on

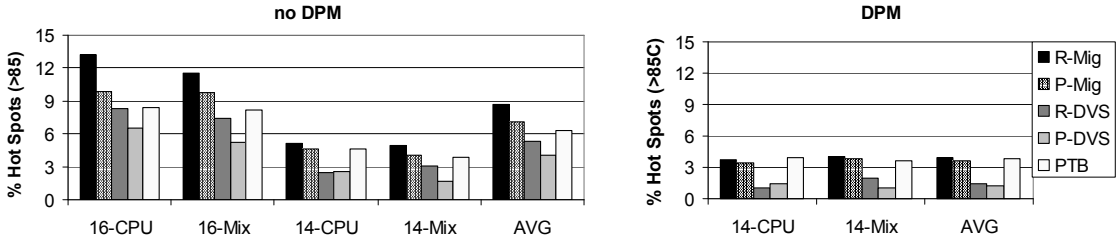

Fig. 20. Thermal Hot Spots (16-core system)

single- and multi-threaded MPSoCs. In our experiments on an UltraSPARC T1 implementation, 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. In addition, we provide a detailed comparison of our ARMA/SPRT based approach to other prediction methods (e.g., exponential moving average, history predictor and least squares predictor).

REFERENCES

[1] Amir H. Ajami, Kaustav Banerjee, and Massoud Pedram. Modeling and analysis of nonuniform substrate temperature effects on global ULSI interconnects. *IEEE Transactions on CAD*, 24(6):849–861, June 2005.

[2] D. Atienza, G. De Micheli, Luca Benini, Jose L. Ayala, P. G. Del Valle, M. DeBole, and V. Narayanan. Reliability-aware design for nanometer-scale devices. In *ASPDAC*, 2008.

[3] D. Atienza, P. Del Valle, G. Paci, F. Poletti, L. Benini, G. De Micheli, and Jose M. Mendias. A fast HW/SW FPGA-based thermal emulation framework for multi-processor system-on-chip. In *DAC*, 2006.

[4] N. L. Binkert, E. G. Hallnor, and S. K. Reinhardt. Network-oriented full-system simulation using M5. In *Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, 2003.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, pages 83–94, 2000.

[6] J. Chang and G. S. Sohi. Cooperative cache partitioning for chip multiprocessors. In *In Proc. of the 21st ACM International Conference on Supercomputing*, 2007.

[7] A. K. Coskun, T. Rosing, and K. Whisnant. Temperature aware task scheduling in MPSoCs. In *DATE*, 2007.

[8] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *ISCA*, 2006.

[9] 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 *ASPLOS*, 2004.

[10] K. Gross, K. Whisnant, and A. Urmanov. Electronic prognostics through continuous system telemetry. In *MFPT*, pages 53–62, April 2006.

[11] K. C. Gross and K. E. Humenik. Sequential probability ratio test for nuclear plant component surveillance. *Nuclear Technology*, 93(2):131–137, Feb 1991.

[12] Jingcao Hu and Radu Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *DATE*, 2004.

[13] W-L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Thermal-aware task allocation and scheduling for embedded systems. In *DATE*, 2005.

[14] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 359–370, 2006.

[15] Failure mechanisms and models for semiconductor devices, JEDEC publication JEP122C. http://www.jedec.org.

[16] A. Karlin, M. Manesse, L. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 1994.

[17] H. Kufluoglu and M. A. Alam. A computational model of NBTI and hot carrier injection time-exponents for MOSFET reliability. *Journal of Computational Electronics*, 3 (3):165–169, Oct. 2004.

[18] A. Kumar, L. Shang, L.-S. Peh, and N. K. Jha. HybDTM: a coordinated hardware-software approach for dynamic thermal management. In *DAC*, pages 548–553, 2006.

[19] E. Kursun, C-Y. Cher, A. Buyuktosunoglu, and P.Bose. Investigating the effects of task scheduling on thermal behavior. In *TACS*, 2006.

[20] 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. *ISSCC*, 2006.

[21] Jinfeng Liu, Pai H. Chou, Nader Bagherzadeh, and Fadi Kurdahi. Power-aware scheduling under timing constraints for mission-critical embedded systems. In *DAC*, 2001.

[22] R. McDougall, J. Mauro, and B. Gregg. Solaris Performance and Tools. *Sun Microsystems Press*, 2006.

[23] M. Mutyam, F. Li, V. Narayanan, M. Kandemir, and M. J. Irwin. Compiler-directed thermal management for VLIW functional units. In *LCTES*, pages 163–172, 2006.

[24] P. Rong and M. Pedram. Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system. In *ASPDAC*, 2006.

[25] T. S. Rosing, K. Mihic, and G. De Micheli. Power and reliability management of SoCs. *IEEE Transactions on VLSI*, 15(4), April 2007.

[26] 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 *DATE*, 2006.

[27] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. Larrabee: a many-core x86 architecture for visual computing. In *ACM SIGGRAPH*, pages 1–15, New York, NY, USA, 2008. ACM.

[28] T. Sherwood, G. Hamerly E. Perelman, and B. Calder. Automatically characterizing large scale program behavior. In *ISCA '02: In 10th International Conference on Architectural Support for Programming*, 2002.

[29] K. Skadron, M.R. Stan, W. Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *ISCA*, 2003.

[30] SLAMD Distributed Load Engine. www.slamd.com.

[31] J.E. Smith. Characterizing computer performance with a single number. In *Communication of ACM, 31(10)*, 1988.

[32] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The case for lifetime reliability-aware microprocessors. In *ISCA*, 2004.

[33] K. Stavrou and P. Trancoso. Thermal-aware scheduling for future chip multiprocessors. *EURASIP Journal on Embedded Systems*, 2007.

[34] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif. Full-chip leakage estimation considering power supply and temperature variations. In *ISLPED*, 2003.

[35] David Tarjan, Shyamkumar Thoziyoor, and Norman P. Jouppi. CACTI 4.0. Technical Report HPL-2006-86, HP Laboratories Palo Alto, 2006.

[36] M. Tremblay and S. Chaudhry. A third-generation 65nm 16-core 32-thread plus 32-scout-thread CMT SPARC processor. In *IEEE International Solid State Circuits Conference (ISSCC)*, 2008.

[37] A. Wald and J. Wolfowitz. Optimum character of the sequential probability ratio test. *Ann. Math. Stat.*, 19:326, 1948.

[38] Q. Wu, M. Martonosi, D. W. Clark, V. J. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks. A dynamic compilation framework for controlling microprocessor energy and performance. In *MICRO 38: Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, pages 271–282, 2005.

[39] Inchoon Yeo, Chih Chun Liu, and Eun Jung Kim. Predictive dynamic thermal management for multicore systems. In *Design Automation Conference*, pages 734–739, June 2008.

[40] Y. Zhang, X. S. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *DAC*, 2002.