Proactive Temperature Balancing for Low Cost Thermal Management in MPSoCs

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

***Abstract*— Designing thermal management strategies that reduce the impact of hot spots and on-die temperature variations at low performance cost is a very significant challenge for multiprocessor system-on-chips (MPSoCs). In this work, we present a proactive MPSoC thermal management approach, which predicts the future temperature and adjusts the job allocation on the MPSoC to minimize the impact of thermal hot spots and temperature variations without degrading performance. In addition, we implement and compare several reactive and proactive management strategies, and demonstrate that our proactive temperature-aware MPSoC job allocation technique is able to dramatically reduce the adverse effects of temperature at very low performance cost. We show experimental results using a simulator as well as an implementation on an UltraSPARC T1 system.**

I. Introduction

The number of CPU cores integrated on a single die continue to increase, elevating the power densities and cooling costs. Providing efficient and low cost thermal management techniques for such multiprocessor system-on-chips (MPSoCs) is one of the most significant challenges, due to the adverse affects of thermal hot spots and large temperature variations. To date, temperature induced reliability, performance and design issues have been managed by design-time optimization and dynamic thermal management. These techniques are not adequate in preventing all the temperature induced problems, especially in deep submicron circuits where the power densities continue to increase. Moreover, conventional dynamic thermal management techniques come at a performance cost. In this work, we propose a proactive and adaptive thermal management technique for multiprocessor system-on-chips (MPSoCs). Our technique predicts the future temperature, and dynamically adjusts the job allocation on the MPSoC to minimize the impact of thermal hot spots and temperature variations without degrading performance.

High temperatures and large temperature variations cause a number of challenges for MPSoCs. In every new generation of chips, the cooling costs dramatically increase. 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 an impact on performance since the effective operating speed of devices decreases with high temperatures. Because of the performance variations caused by temperature differences, global clock networks are especially vulnerable to such spatial variations. 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 [11]. Large spatial temperature gradients accelerate the parametric reliability problems caused by negative bias temperature instability (NBTI) and hot carrier injection (HCI) [13]. Temporal temperature fluctuations, i.e. high magnitude and frequency of thermal cycles, cause package fatigue and plastic deformations [11].

Most of the previously proposed thermal management techniques focus on maintaining the temperature below critical levels. Methods such as clock gating and temperature-triggered frequency scaling [22] prevent hot spots by responding to a given temperature threshold. Obviously, such techniques maintain the temperature below a critical level at a performance cost. In the multiprocessor domain, techniques such as thread migration and applying PID control for keeping the temperature stable [5] 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 the temperature reaches a certain level. 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 [20]. Even though some dynamic temperature aware MPSoC scheduling techniques (e.g. [4]) are able to reduce thermal variations as well as hot spots in comparison to conventional thermal or power management, they are still reactive in nature, and therefore take action after undesirable thermal profiles are observed.

In this work, we propose a proactive thermal management method for MPSoCs to prevent thermal problems before they occur 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. 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 [27]. Early detection of variations enable us to update the ARMA model for the current thermal dynamics immediately, and avoid inaccuracy.

Utilizing the ARMA temperature predictor, 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 a simulator based on an UltraSPARC T1 [16], and also on a real system implementation. We use real life workloads as measured by the Continuous System Telemetry Harness (CSTH) [7]. Our proactive temperature-aware allocation strategy reduces the frequency of hot spots and large gradients significantly in comparison to reactive thermal management strategies, without affecting performance.

To summarize, the contributions of this work are:

- We demonstrate how to use ARMA modeling to predict future temperature and guide proactive thermal management.
- We utilize sequential probability ratio test (SPRT) to monitor temperature dynamics at runtime and adapt the ARMA predictor if the workload dynamics vary. 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 pre-

dicted 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.

We start by discussing the related work in Section II. In Section III we provide the details of the ARMA based learning / prediction method, and Section IV discusses the online adaptation framework. In Section V, we explain all the thermal management techniques we study in this work, including our proactive MPSoC thermal management technique. Section VI provides the experimental methodology and results, and we conclude in Section VII.

II. Related Work

In this section, we discuss the prior work in multicore scheduling, and also energy and thermal management. Optimizing multicore scheduling with energy and performance (or timing) constraints has been studied quite extensively in the literature. A power management strategy for mission critical systems containing heterogeneous devices is proposed in [17]. A static scheduling method optimizing concurrent communication and task scheduling for heterogeneous network-on-chips is proposed in [9]. 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 [19]. In [21], the 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 dynamic voltage scaling (DVS) has been formulated using a two-phase framework in [28].

As power-aware policies are not always sufficient to prevent temperature induced problems, thermal modeling and management methods have been proposed. HotSpot [22] 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 [10]. RAMP [25] provides a reliability model at the architecture level for temperature related failures, and optimizes the architectural configuration and power/thermal management policies for reliable design. In [20], 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 [22]. Heat-and-Run performs temperature-aware thread assignment and migration for multicore multithreaded systems [6]. Kumar et al. propose a hybrid method that coordinates clock gating and software thermal management techniques such as temperature-aware priority management [14]. The multicore thermal management method introduced in [5] combines distributed DVS with process migration. The temperature-aware task scheduling method proposed in [4] achieves better thermal profiles than conventional thermal management techniques without introducing a noticeable impact on performance. Predictive thermal management for multimedia applications has been introduced in [24], where the applications are profiled offline to determined the safe frequencies to run them.

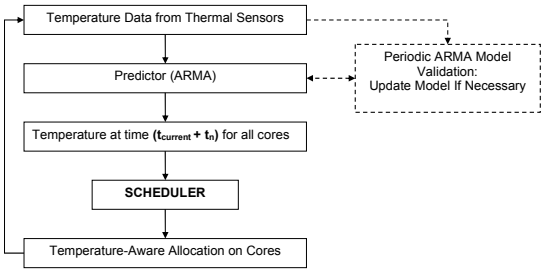

Fig. 1. Flow Chart of the Proposed Technique

Our goal in this work is to introduce a proactive and adaptive temperature management method for MPSoCs. The key difference to previous work is that our technique utilizes temperature predictions at runtime. As opposed to taking action after temperature reaches a certain level, our technique estimates the hot spots and temperature variations in advance, and dynamically modifies the job allocation to minimize temperature's adverse effects. The prediction technique does not rely on offline profiling; thus, it can be utilized for a variety of applications and architectures. We also provide a comprehensive comparison of various examples of reactive and proactive thermal management techniques in not only simulation but also on a real implementation.

III. Accurate Temperature Prediction

In this section, we provide an overview of our proactive temperature management approach, and explain the methodology for accurate temperature prediction at runtime. Temperature prediction is a critical part in our proactive temperature management technique, which adjusts the workload distribution on an MPSoC using the predictions to avoid the occurrence of hot spots and temperature variations. Our approach anticipates and avoids thermal problems before they occur, as opposed to reacting to hot spots and variations after they appear on the system. This way, we can reduce the temperature induced problems much more effectively at negligible performance cost.

Figure 1 provides an overview of our technique. Current chips typically contain several thermal sensors, which can be read by a Continuous System Telemetry infrastructure for collecting and analyzing time series sensor data [7]. 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 IV.

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 stationary (in the stochastic sense), 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. As we adapt the model when the ARMA model no longer fits the workload, 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, 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.

During identification, we use an automated trial and error strategy. We start with the simplest model, i.e. ARMA(1,0), and increase the model order at every iteration, and measure the "goodness-of-fit". We use *Final Prediction Error (FPE)* to evaluate the goodness-of-fit of the models. 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.

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

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.

After computing the ARMA model, we test its prediction capabilities. This is achieved by predicting a portion of the time series data which has been previously set aside for testing the model (i.e. not used for training the ARMA model). 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 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.

As an example, we have applied the ARMA prediction methodology to a sample temperature trace. The trace is obtained through HotSpot [22] for a web server workload running on a system with

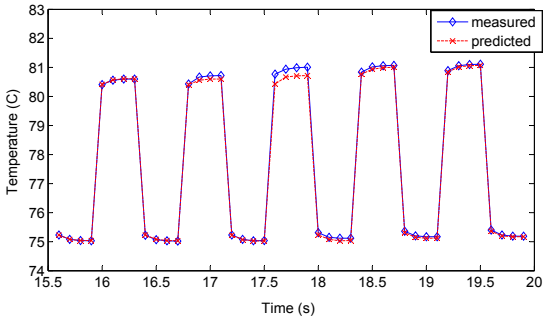

Fig. 2. Temperature Prediction

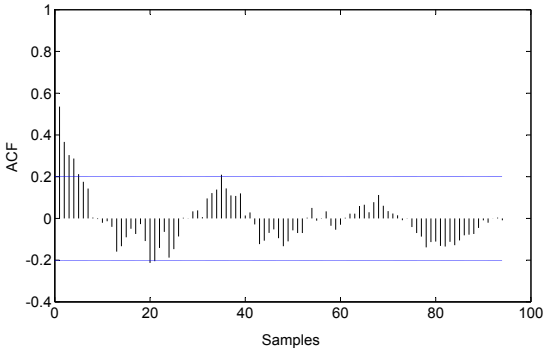

Fig. 3. Autocorrelation Function of the Residuals

a thermal management policy that swaps workload among hot and cold cores periodically. 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 data, we formed ARMA models with $FPE << 1$. We saved the last 50 samples of the data in both experiments 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. In Figure 2, we observe frequent cycles in the trace due to an allocation policy that swaps jobs between hot and cool cores. The prediction almost matches the observed values. 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 dashed 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 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 upto 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 further reduce the overhead significantly.

In Figures 4 and 5, 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 4.

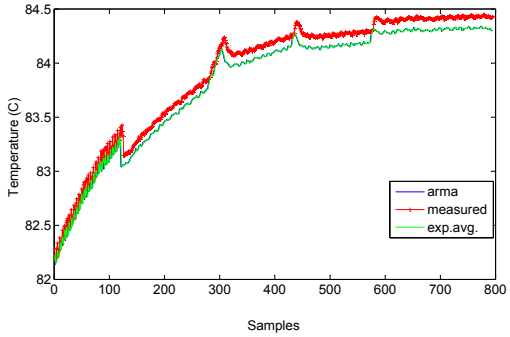

Fig. 4. Comparison of Predictors - Stable Temperature

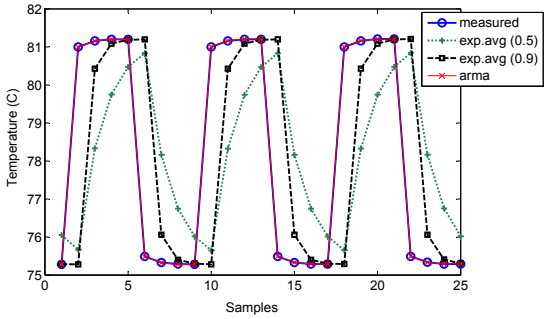

Fig. 5. Comparison of Predictors - Thermal Cycling

However, when there are rapid temperature changes, such as thermal cycling, exponential average predictor performs poorly, such as in Figure 5. 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 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.

IV. 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 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 [27], 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 as in the standard deviation method described in the previous section, 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. The residuals of an ARMA model with good prediction capabilities should be random and fluctuate close to zero. 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. 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 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 [8], 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 [27], 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.

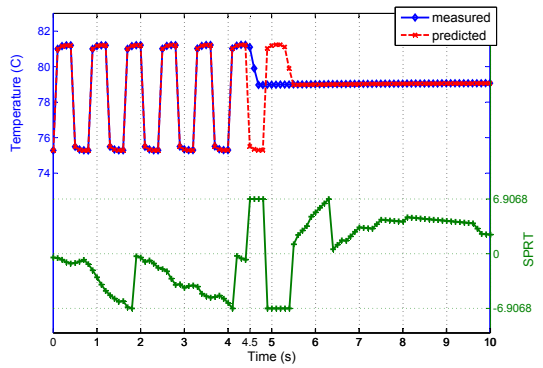

Fig. 6. Online Detection of Variations in Thermal Characteristics

In Figure 6, we demonstrate a case where the temperature dynamics change, and the SPRT detects this change immediately (see $t = 4.5s$ in the figure). The SPRT thresholds of ± 6.9068 correspond to A and B values of 0.001. When $SPRT >= 6.9068$ 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 using a 10% threshold. While the standard deviation method quickly detects the change in temperature dynamics in the case of abrupt changes as in Figure 6, 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.

V. 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 (DVFS)`. 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 [12]. The DVFS 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. [6]). 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 DVFS` reduces the voltage/frequency (V/f) setting on a core if the threshold temperature is exceeded, similar to the frequency scaling approach in [22]. 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. 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 DVFS` 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.

VI. Experimental Results

Our experimental results are based on the UltraSPARC T1 processor [16]. The average power consumption including leakage and area distribution of the units on the chip are provided in Table I, and the floorplan is available in [16].

TABLE I. 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

In this work, we show two sets of experimental results. In the first set, we provide results from the experiments run on our simulator, where the simulator set-up is based on the power and thermal characteristics of UltraSPARC T1. In the second set of experimental results, we implemented the policies in Solaris and ran the workload on the UltraSPARCT1 in real time.

In our simulations, we leveraged the Continuous System Telemetry Harness (CSTH) [7] to gather detailed workload characteristics of real applications. We sampled the utilization percentage for each hardware thread at every second using `mpstat` [18]. 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` [18].

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 [23] 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 II. 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 II. 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 [16], so we assumed that the instantaneous power consumption is equal to the average power at each state (active, idle, sleep). 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, we used the second-order polynomial model proposed in [26]. 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 [22] 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.

In our 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.[15]).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 [11]. 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.

We next 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.

First we provide the results obtained through 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 DVFS cannot eliminate all the hot spots, especially for workloads with medium to high utilization level. Performing migration or DVFS 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 DVFS 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 7 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 8 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

TABLE III. PERCENTAGE OF THERMAL HOT SPOTS AND PERFORMANCE COMPARISON

	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

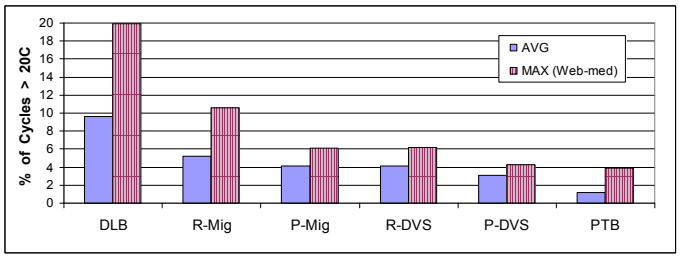

Fig. 7. Temperature Cycles - with DPM (Simulator)

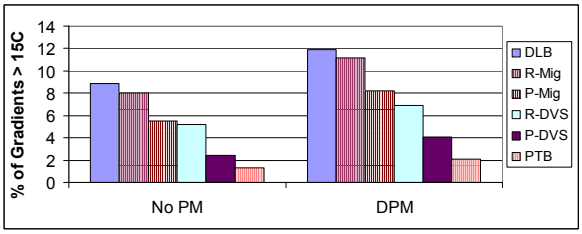

Fig. 8. Spatial Gradients (Simulator)

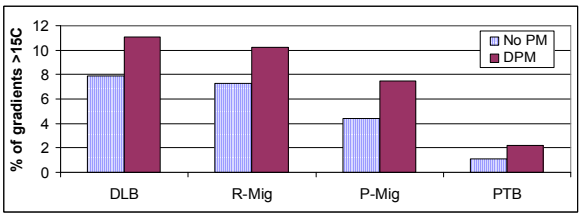

Fig. 9. Spatial Gradients (Implementation)

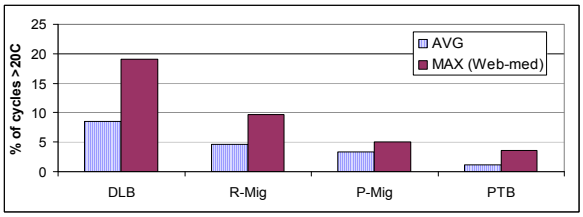

Fig. 10. Thermal Cycles - with DPM (Implementation)

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.

TABLE IV. TEMPERATURE RESULTS FOR COMBINED WORKLOADS

	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

Implementation on UltraSPARC T1

In addition to the simulator, 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 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 9 and 10, 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

TABLE V. HOT SPOTS ON THE 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

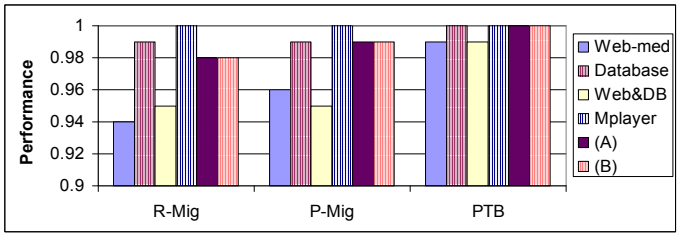

Fig. 11. Normalized Performance (Implementation)

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 11 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.

VII. CONCLUSION

In this paper, we have presented a proactive temperature management approach for multiprocessor system-on-chips (MPSoCs). We utilize autoregressive moving average (ARMA) modeling to accurately predict future temperature on each core based on the history of temperature telemetry. 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. 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.

ACKNOWLEDGEMENT

This work has been funded by Sun Microsystems, and the University of California MICRO grant 06-198.

REFERENCES

[1] A. H. Ajami, K. Banerjee, and M. 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. D. Micheli, L. Benini, J. L. Ayala, P. G. D. Valle, M. DeBole, and V. Narayanan. Reliability-aware design for nanometer-scale devices. In *ASPDAC*, 2008.

[3] 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 *DAC*, 2006.

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

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

[6] 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.

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

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

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

[10] 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.

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

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

[13] 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.

[14] 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.

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

[16] 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.

[17] J. Liu, P. H. Chou, N. Bagherzadeh, and F. Kurdahi. Power-aware scheduling under timing constraints for mission-critical embedded systems. In *DAC*, 2001.

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

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

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

[21] 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.

[22] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *ISCA*, 2003.

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

[24] J. Srinivasan and S. V. Adve. Predictive dynamic thermal management for multimedia applications. In *ICS*, 2003.

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

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

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

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