Proactive Temperature Management in MPSoCs

Ayse Kivilcim Coskun[†] Tajana Simunic Rosing[†] Kenny C. Gross[‡]
[†]University of California, San Diego [‡]Sun Microsystems, San Diego
{acoskun, tajana}@ucsd.edu kenny.gross@sun.com

ABSTRACT

Preventing thermal hot spots and large temperature variations on the die is critical for addressing the challenges in system reliability, performance, cooling cost and leakage power. Reactive thermal management methods, which take action after temperature reaches a given threshold, maintain the temperature below a critical level at the cost of performance, and do not address the temperature variations. In this work, we propose a proactive thermal management approach, which estimates the future temperature using regression, and allocates workload on a multicore system to reduce and balance the temperature to avoid temperature induced problems. Our technique reduces the hot spots and temperature variations significantly in comparison to reactive strategies.

Categories and Subject Descriptors: B.8 [**Performance and Reliability**]: General; C.4 [**Computer Systems Organization**]: Performance of Systems.
General Terms: Management, Design, Reliability.
Keywords: Thermal Management, Multiprocessor, ARMA.

1. INTRODUCTION

The increasing performance demands and technology scaling have significantly elevated the power density of chips, resulting in thermal hot spots as well as large spatial and temporal 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 in the deep submicron era, especially the challenges caused by temperature variations. Moreover, conventional dynamic thermal management techniques come at a performance cost. In this work, we propose a proactive thermal management technique for multiprocessor system-on-chips (MPSoCs). Our technique continuously predicts the temperature several time increments into the future, and dynamically adjusts job allocation on the MPSoC to minimize the impact of thermal hot spots and temperature variations without degrading performance.

Hot spots and temperature variations cause a number of challenges. First, the cooling costs dramatically increase with every generation of chips due to high temperatures. In addition to hot spots, spatial thermal gradients on the die affect the cooling cost as large gradients decrease cooling efficiency. Second, leakage is exponentially related to temperature, so increasing temperatures increase leakage power. Third, as the effective operating speed of devices decreases with higher temperatures, temperature has an impact on runtime performance and also performance predictability at design time. 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 temperature gradients in time and space adversely affect reliability. Failure mechanisms such as electromigration, stress migration, and dielectric breakdown, which cause permanent device failures, are exponentially dependent on temperature [10]. Large spatial temperature gradients accelerate the parametric reliability problems caused by negative bias temperature instability (NBTI) and hot carrier injection (HCI) [11]. Temporal temperature fluctuations, i.e. high magnitude and frequency of thermal cycles, cause package fatigue and plastic deformations, and lead to permanent failures as well [10].

The majority of the thermal management techniques proposed previously, such as clock gating or temperature triggered frequency scaling [19], maintain the temperature below a given threshold at a performance cost. In the multiprocessor domain, techniques such as thread migration and applying PID control [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 temperature reaches a certain level. Therefore, they do not target reducing the temperature as much as possible to minimize the impact on reliability and to ease the design complexity issues caused by temperature. In addition, conventional dynamic thermal management techniques do not focus on balancing the temperature spatially on the chip or temporally, and they can create large thermal gradients or cycles. Reliability degradation can get accelerated because of the cycles created by workload rate changes and power management decisions, especially in systems with dynamic power management (DPM) that turn off cores [17]. Some dynamic temperature-aware MPSoC scheduling techniques (e.g. [4]) are able to reduce temporal and spatial variations as well as hot spots in comparison to conventional thermal or power management for the average case. However they do not guarantee to be effective for all execution periods, considering the trade-off between temperature and performance varies for different workloads.

In this work, we propose a proactive thermal management method for MPSoCs. We utilize autoregressive moving average (ARMA) modeling for estimating future temperature accurately based on a moving window of temperature history. In our experiments, we

ISLPED'08, August 11–13, 2008, Bangalore, India..

have seen that as workload goes through stationary phases, temperature can be estimated accurately by regressing the thermal telemetry. Using the temperature prediction, we preferentially allocate incoming jobs to cores to minimize and balance the temperature across the die. We evaluate several performance-aware and temperature-aware allocation strategies for MPSoCs on an UltraSPARC T1 system [13]. The thermal behavior of our technique is evaluated using 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 by about 60%, spatial gradients by 70% and thermal cycles by 20% in comparison to reactive thermal management strategies, without affecting performance.

2. RELATED WORK

In this section, we briefly discuss the techniques for multicore scheduling and thermal management. 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 [14]. A static scheduling method optimizing concurrent communication and task scheduling for heterogeneous network-on-chips is proposed in [8]. 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 [16]. In [18], 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 [23]. In [15], 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 [19] 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 [9]. RAMP provides a reliability model at architecture level for temperature related intrinsic hard failures [21]. It analyzes the effects of application behavior on reliability, and optimizes the architectural configuration and power/thermal management policies for reliable design. In [17], 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], and as a case study, a reliability-aware register assignment policy is introduced for increasing register file reliability.

Dynamic thermal management controls over-heating by keeping the temperature below a critical threshold. Computation migration and fetch toggling are examples of such techniques [19]. 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 [12]. 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.

Our goal in this work is to introduce a thermally-aware job allocation method for MPSoCs. The key differentiating point with previous work is that our technique is predictive. As opposed to taking action after temperature reaches a certain threshold level, our technique estimates the hot spots and temperature variations in advance, and modifies the job allocation to minimize temperature's adverse effects.

3. PROACTIVE THERMAL MANAGEMENT

Previously proposed thermal management strategies are typically reactive in nature; that is, they stall or slow down cores, or migrate workload when temperature reaches a critical value. In addition to the performance cost they cause, reactive approaches do not always minimize the frequency of hot spots observed. Moreover, conventional thermal management does not address balancing the temperature profile spatially on the die and temporally during execution. In this work, we present a proactive thermal-aware scheduling approach, which adjusts the workload distribution on a multiprocessor SoC to minimize the hot spots and temperature variations. As the approach is predictive, it prevents thermal problems before they occur, as opposed to reacting to hot spots and variations after they appear on the system. This way, we can mitigate temperature induced problems much more effectively at negligible performance cost. In this section, we provide the underpinning mathematical details of our technique.

Figure 1 shows the flow chart 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 a moving history window of temperature measurements, we predict temperature t_n time increments into the future using an autoregressive moving average (ARMA) model. As workload goes through stationary phases and temperatures change slowly (on the order of milliseconds) due to thermal time constants, estimating future temperature based on the regression of previous telemetry values gives accurate results. The scheduler then allocates the threads to cores to balance the temperature distribution across the die. The ARMA model utilized for temperature prediction is derived based on a temperature trace representative of the thermal characteristics of the current workload. During execution, the workload dynamics can change over time. Consequently, the ARMA model parameters need to be adaptive to achieve robustness and maintain high prediction accuracy. To provide runtime adaptation, we monitor the workload through the temperature measurements, validate the ARMA model periodically, and update the model as needed. In this section, we provide the details of the ARMA-based prediction method, how to dynamically adapt to workload variations, and our temperature-aware scheduling technique.

3.1 Temperature Prediction with ARMA

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 the future temperature of cores using the temperature values observed in the past. ARMA modeling assumes the modeled process is stationary in the statistical sense, and that there is some degree of serial correlation in the data. In a stationary process, the probability distribution does not change over

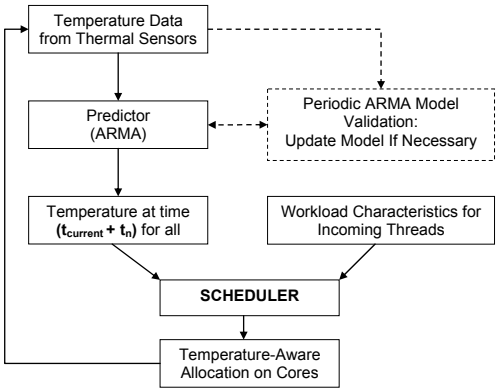

Figure 1: Flow Chart of the Proposed Technique

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 stationarity assumption does not introduce inaccuracy.

Other well-known prediction methods, such as exponential averaging, may work well when the temperature profile is stable. However, ARMA modeling is capable of capturing temperature dynamics even when the thermal profile is highly variant, e.g. when there is thermal cycling.

An ARMA(p,q) model can be 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.

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

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 (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. Taking the number of estimated parameters into account 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 as a function of lag time, and is useful for identifying repeating patterns in a signal if there are any. If model residuals are random, the ACF of all residuals (except for lag zero) fluctuates 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). We observe the *mean squared error (MSE)* of the predicted trace, and ensure that it is close to zero. For example, to tolerate 1^oC of error in average, we would check whether $MSE < 1$, as $MSE < \frac{\sum_i^H {Err_i}^2}{H}$, where H is the length of the data and Err_i is the prediction error for i^{th} data point.

As an example, we have applied the ARMA prediction methodology to a sample temperature trace. The trace (obtained using HotSpot [19]) is 500 samples long, sampled at every 100 ms. Using the first 400 samples of the data, we formed an ARMA(5,0) model with $FPE << 1$. We saved the last 100 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 results of the temperature prediction is demonstrated in Figure 2. The prediction almost perfectly matches the observed values. It should be noted that, for more flat temperature curves, designing an accurate ARMA predictor is even easier.

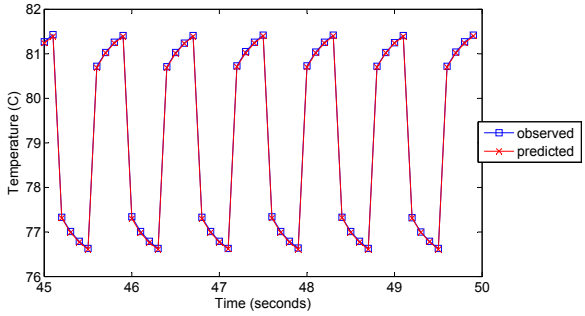

Figure 2: Temperature Prediction

Figure 3 shows the ACF of the residuals in the model. 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 the residuals are random.

3.2 Runtime Adaptation of the ARMA Model

As discussed previously, 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 periodically validating the ARMA model.

To perform online validation, we maintain a history window of temperature on each core. The length of the window is empirically selected based on thermal time constants and workload characteristics. Using the recent temperature history and the current ARMA model, we compute the residuals by differencing the predicted data from the observed data. The ACF of the residuals are investigated,

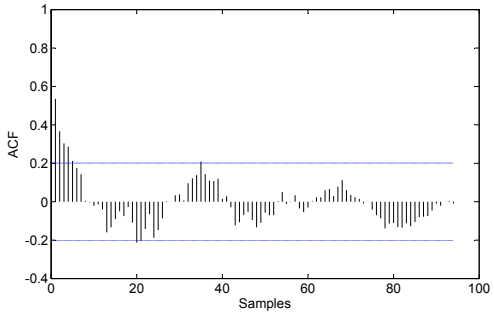

Figure 3: Autocorrelation Function of the Residuals

as previously explained in the validation stage of the model. If the residuals are not random, then we conclude that the current ARMA model does not fit the current workload characteristics well enough.

We also measure the mean squared error (MSE) of prediction. If the error is diverging from zero over time, this is a sign that the current model is not working well any longer. Monitoring the MSE allows for fast detection of inaccurate predictions. When we determine the current ARMA model is not suitable anymore, we update the model on the fly. Until the new model is computed, we continue using the old model.

3.3 Temperature-Aware Allocation

The goal of our temperature-aware workload allocation technique is to minimize the magnitude and frequency of the thermal hot spots and temperature variations. To achieve this goal, we evaluate several techniques, and propose a predictive temperature-aware job allocation method for MPSoCs.

In the system model we assume for the multicore system, each core is associated with a *dispatching queue*, which keeps the threads allocated to the core. This is the typical set-up in modern multicore OS schedulers (e.g. Solaris), which are based on multilevel queuing. The dispatcher allocates the incoming threads to queues based on the policy. Further balancing (i.e. migration) among the queues are possible depending on the policy requirements. 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. Next we explain the details of all the scheduling techniques we compare in this work.

Performance-Aware Dynamic Load Balancing (Default policy - DLB): The conventional dispatching method used in multicore schedulers is to assign 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.

Performance-Aware Balancing Combined With Reactive Thermal Management: One way to perform temperature-aware allocation is to initially utilize performance-aware balancing while dispatching jobs, then at runtime migrate threads from cores that reach a critical threshold. This strategy is an example of reactive thermal management (e.g. [6]), which takes action after a certain temperature threshold is reached.

Proactive Temperature-Aware Load Balancing: In this technique, the dispatcher follows the principle of locality (i.e. running the threads on the same core they ran before) during initial assignment. The ARMA prediction method is applied at runtime, and if the temperature of cores are projected to have an 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. Note that we move the threads waiting in queue, so migration does not include stalling the running thread. Moving the waiting threads in the queues is already performed by the default policy for load balancing purposes, so this technique does not introduce additional overhead.

4. EXPERIMENTAL RESULTS

Our experimental results are based on the UltraSPARC T1 processor [13]. The average power consumption including leakage and area distribution of the units on the chip are provided in [4], and the floorplan is available in [13].

In our simulations, we leveraged the Continuous System Telemetry Harness (CSTH) [7] to gather workload characteristics of real applications. We sampled the utilization percentage for each hardware thread at every second using `mpstat`. We recorded half-hour long traces for each benchmark. To determine the active/idle time slots of cores accurately, we recorded the length of user and kernel threads using `DTrace`.

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 [20] 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 1. The utilization ratios are averaged over all cores throughout the execution. Using `cpustat`, we recorded the cache misses and floating point (FP) instructions per 100K instructions.

Table 1: 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 [13], 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 [22]. We determined the coefficients in the model empirically to match the normalized leakage values in [22]. We scaled the leakage power at the default voltage level and nominal temperature based on this model considering the change in voltage state and temperature. 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 [19] for thermal modeling, and modified the floorplan and thermal package characteristics for UltraSPARC T1. We ran the thermal simulations with a sampling interval of 100 ms, which provided good precision. We initialized HotSpot with steady state temperature values.

We next evaluate the techniques we have discussed in the previous section. We implemented **Dynamic Power Management (DPM)** and **Dynamic Voltage Scaling (DVS)** to demonstrate how the allocation strategies behave with power management. **DPM** turns off idle cores based on a fixed timeout strategy. **DVS** applies a dynamic voltage/frequency scaling policy which reduces the V/f level depending on the utilization observed on each core in the last interval. As a reactive temperature management policy, we implemented temperature triggered thread migration. **Migration** migrates threads from hot cores to cooler cores when a temperature threshold is exceeded. Note that such a policy is different than balancing the threads waiting in the dispatch queues.

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 [4].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 [10]. 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.

Table 2 shows a detailed analysis of the hot spots observed on the system. *DLB* represents the performance-aware dynamic load balancing strategy, *Migration* stands for the policy combining temperature-triggered migration with DLB, and *Proactive* is our proactive temperature-aware allocation technique. We demonstrate 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), DPM and DVS. We observe that, migration of workload upon reaching the threshold cannot eliminate all the hot spots, especially for workloads with medium to high utilization level. The threshold to trigger migration can be set to a lower level to reduce the hot spots further at the cost of higher performance degradation.

DPM and DVS reduces the thermal hot spots to some extent as they reduce the temperature when the system has idle time. However, performing proactive temperature management results in the best thermal profile among these techniques, as demonstrated in the table.

Figure 4 shows the average percentage of time thermal cycles above 20^oC were observed. We also plotted the workload with the maximum thermal cycling, i.e. Web-med. We only consider the case with DPM for the thermal cycling results, as putting cores into the sleep state creates larger cycles. Our technique achieves very significant reduction in thermal cycles, as it continuously balances the workload among the cores according to their expected temperature. As migration reacts to temperature thresholds, it cannot avoid the temperature imbalance in time as much as the proactive technique.

In Figure 5, we demonstrate the average percentage of time large spatial gradients above 15^oC occurred while running the policies.

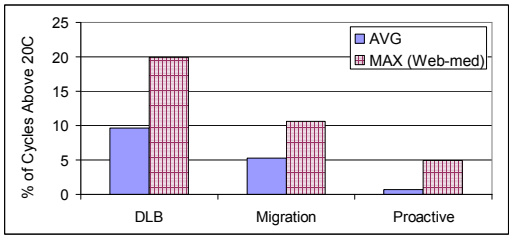

Figure 4: Temperature Cycles (with DPM)

DPM creates larger gradients due to the low temperatures on the cores that go into the sleep state. Proactive thermal management can reduce the gradients to below 4% in average, and to less than 2% for the case with DVS. By reducing the frequency and voltage on the cores, DVS achieves lower amount of temperature variations on the die.

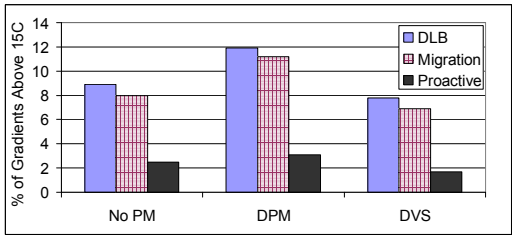

Figure 5: Spatial Gradients

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 3, we show the results for running the following combinations of workload with the *Proactive* 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.

Table 3: Thermal Results for Combined Workloads

	Hot Spots (%)	Cycles (%)	Gradients (%)
Web-med	4.8	4.9	5.5
Web&DB	4.6	3.1	6.7
Mplayer	0	0.5	0.9
(A) Web-med, Web&DB	4.9	4.1	6.0
(B) MPlayer, Web-med	2.9	3.1	3.5

If the ARMA model no longer adequately fits the current workload, we compute a new model while we continue using the old ARMA model during the adaptation period. Computing the new ARMA model is a fast process. 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. Thus, the model will be inaccurate for only on the order of several time constants. The results in Table 3 confirm this observation, as the combined workload slightly changes the temperature statistics.

Table 2: Percentage of Thermal Hot Spots

	no PM			DPM			DVS		
Workload	DLB	Migration	Proactive	DLB	Migration	Proactive	DLB	Migration	Proactive
Web-med	25.9	12.9	6.9	19.5	10.9	4.8	21.2	7.7	2.8
Web-high	39.1	22.1	9.5	37.4	21.6	9.3	37.5	19.2	7.2
Database	8.3	2.1	0.7	4.6	1.5	0.5	3.8	1.5	0
Web&DB	32.4	15.3	5.3	27.8	13.2	4.6	24.3	10.7	2.1
gcc	7.2	1.8	0.4	3.8	1.3	0.2	3.1	0.5	0
gzip	2.9	0.6	0.0	1.3	0.5	0	1.5	0.1	0
Mplayer	4.9	0.7	0.0	1.7	0.5	0	1.6	0	0
Mplayer&Web	13.3	9.4	2.1	8.9	7.2	1.4	6.2	4.9	1
AVG	16.8	8.1	3.1	13.1	7.1	2.6	12.4	5.6	1.6

5. CONCLUSION

In this paper, we have proposed a proactive temperature-aware workload allocation technique for MPSoCs. Our technique continuously predicts the future temperature on each core in the MPSoC using autoregressive moving average (ARMA) modeling as applied to a moving history window of temperature telemetry. Based on the predictions, the threads waiting in dispatch queues of cores are proactively balanced by preferentially shifting threads from hotter cores to cooler ones. To adapt to runtime changes in workload, the automated algorithm periodically evaluates how closely the current ARMA model fits the recent temperature dynamics, and updates the model whenever necessary. Our technique achieves significant reduction in the frequency of thermal hot spots and temperature variations with respect to performance-aware workload allocation or reactive thermal management at negligible performance cost.

Acknowledgment

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

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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] P. Rong and M. Pedram. Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system.

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

[18] 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.

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

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

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

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

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