Reinforcement Learning-Based Inter- and Intra-Application Thermal Optimization for Lifetime Improvement of Multicore Systems

Anup Das
National University of Singapore
Singapore - 117583
akdas@nus.edu.sg

Rishad A. Shafik
University of Southampton
Southampton SO17 1BJ, UK
rishad.shafik@ecs.soton.ac.uk

Geoff V. Merrett
University of Southampton
Southampton SO17 1BJ, UK
gvm@ecs.soton.ac.uk

Bashir M. Al-Hashimi
University of Southampton
Southampton SO17 1BJ, UK
bmah@ecs.soton.ac.uk

Akash Kumar
National University of Singapore
Singapore - 117583
akash@nus.edu.sg

Bharadwaj Veeravalli
National University of Singapore
Singapore - 117583
elebv@nus.edu.sg

ABSTRACT

The thermal profile of multicore systems vary both within an application's execution (intra) and also when the system switches from one application to another (inter). In this paper, we propose an adaptive thermal management approach to improve the lifetime reliability of multicore systems by considering both inter- and intra-application thermal variations. Fundamental to this approach is a reinforcement learning algorithm, which learns the relationship between the mapping of threads to cores, the frequency of a core and its temperature (sampled from on-board thermal sensors). Action is provided by overriding the operating system's mapping decisions using affinity masks and dynamically changing CPU frequency using in-kernel governors. Lifetime improvement is achieved by controlling not only the peak and average temperatures but also thermal cycling, which is an emerging wear-out concern in modern systems. The proposed approach is validated experimentally using an Intel quad-core platform executing a diverse set of multimedia benchmarks. Results demonstrate that the proposed approach minimizes average temperature, peak temperature and thermal cycling, improving the mean-time-to-failure (MTTF) by an average of 2x for intra-application and 3x for inter-application scenarios when compared to existing thermal management techniques. Furthermore, the dynamic and static energy consumption are also reduced by an average 10% and 11% respectively.

1. INTRODUCTION

A major challenge of modern multicore systems is decreasing lifetime reliability, threatened by high power densities and hence elevated operating temperatures. This leads to an acceleration of device wear-out. Thermal management has attracted significant attention both in industry and academia. Examples include dynamic thermal management using voltage and frequency scaling [7], slack time management [10], peak temperature management through system-level task scheduling [3] and thermal stress management through application task mapping [2] (refer to Section 2 for a summary of related works). These approaches, however, suffer from the following limitations.

First, modern multicore systems switch between applications exhibiting wide performance and workload variations, and therefore the thermal behavior of these systems vary both within (intra) and across (inter) applications. Although intra-application thermal variations are considered in existing studies, inter-application variations are not addressed. Second, the existing studies focus on average temperature reduction; thermal cycling is not accounted. Last, the existing adaptive techniques are either implemented on a simulator or rely on time-consuming thermal prediction from the *HotSpot* tool [14], limiting their scalability.

DAC '14, June 01 - 05 2014, San Francisco, CA, USA

http://dx.doi.org/10.1145/2593069.2593199.

In this paper, we address the above gaps and present a dynamic thermal management approach for multicore systems that adapts to thermal variations within (intra) and across (inter) applications. Fundamental to this approach is a run-time system, which interfaces with the on-board thermal sensors and uses reinforcement learning to learn the relationship between the mapping of threads to cores, the frequency of a core and its temperature. The aim is to control the average temperature and the thermal cycling to achieve an extended mean time to failure (MTTF). This paper makes the following contributions:

1. inter-and intra-application thermal management using thread-to-core allocation (through CPU affinity) and dynamic frequency control (through CPU governors)[1];
2. separation of the temperature sampling interval from the decision interval of the conventional reinforcement learning algorithm to accurately model (and hence control) the average temperature and thermal cycling; and
3. implementation of the run-time system incorporating the machine learning algorithm on a real platform.

The proposed approach is implemented on an Intel quad-core platform running Linux kernel 3.8.0. A set of multimedia applications from the *ALPBench* suite [11] are executed on the platform. Results demonstrate that the proposed approach minimizes average temperature and thermal cycling, leading to a significant improvement in MTTF as demonstrated in Section 6.

The remainder of this paper is organized as follows. A brief introduction to related works is presented in Section 2 and the motivation in Section 3. This is followed by the preliminaries on reliability in Section 4 and an overview of the reinforcement learning-based approach in Section 5. Evaluation of the proposed technique is presented next in Section 6 and the paper is concluded in Section 7.

2. RELATED WORKS

The existing studies on thermal optimization can be classified into two categories – static and dynamic. Static techniques determine application mapping and scheduling offline to minimize peak temperature or thermal cycles [17]. Dynamic techniques optimize temperature at run-time. Since this work belongs to the latter category, dynamic thermal management techniques are discussed in depth. A slack borrowing technique is proposed in [10] to dynamically manage peak temperature for MPEG-2 decoder. A reinforcement-learning based adaptive technique is proposed in [3] to optimize temperature by controlling task mapping based on the temperature of the current iteration. A neural network based adaptive technique is proposed in [9] to reduce peak temperature. Both these techniques rely on the *HotSpot* tool for temperature prediction. The relationship between temperature and voltage and frequency of operation is formulated in [8, 12]. Based on this, an online heuristic is proposed to determine the voltage and frequency of the cores to minimize the temperature. This technique does not perform learning, resulting in re-performing the same optimization for the same environment. A reinforcement learning algorithm is proposed in [7] to manage performance-thermal

[1]CPU affinity enables the binding of a thread of an application to a physical core or a range of cores.
CPU governors are power schemes for the CPU deciding the frequency of operation of the cores.

Table 1. Summary of related Works

Related Works	Machine Learning	Inter-Application	Thermal Cycling	Temperature Measurements	Platform
[6]	√	×	×	thermal gun	*FPGA*
[3]	√	×	√	*HotSpot*	multicore
[2]	×	×	√	*HotSpot*	simulation
[9]	√	×	×	*HotSpot*	simulation
[8, 12]	×	√	×	thermal model	simulation
[7]	√	×	×	sensors	multicore
Proposed	√	√	√	sensors	multicore

trade-offs by sampling temperature data from the on-board thermal sensors. A distributed learning agent is proposed in [6] to optimize peak temperature with a given power budget. The technique is implemented on an FPGA with temperature measurement using an external thermal gun.

Table 1 summarizes the related works. As can be seen, some of the existing studies rely on the *HotSpot* tool for temperature prediction. Although the *HotSpot* tool can be used to validate static thermal management policies, the high simulation time of the tool can potentially cause deadline misses for dynamic thermal management of real-time systems. In some of the existing works, temperature is estimated by solving a RC equivalent thermal model. These models are usually complex and difficult to solve using direct mathematical techniques such as LU decomposition. Thermal guns have also been used to measure the temperature; however, this suffers from limited accuracy, especially to capture thermal cycling. Furthermore, deploying thermal guns for real system is not practical. Finally, using performance counter for estimating temperature can lead to inaccuracy in the thermal values. These motivate the use of thermal sensors, even at the expense of limited speed of operation, and as such [7] is used to compare the proposed approach.

3. MOTIVATIONAL EXAMPLE

Thermal management using voltage and frequency control is already demonstrated in prior works (e.g. [7]). To establish the importance of thread allocation on the thermal behavior of applications, an experiment is conducted on an Intel quad-core platform by executing two multi-threaded (6 threads) applications (*face recognition* and *mpeg2 encoding*) back-to-back. The thermal profiles obtained using Linux's default thread-to-core allocation and scheduling is shown in red in Figure 1. From the thermal profiles it can be seen that *face recognition* is characterized by a higher average temperature with lower thermal cycling leading to peak temperature related reliability issues such as electro-migration (EM) and negative bias temperature instability (NBTI). Application *mpeg2 encoding* on the other hand exhibits lower average temperature with higher thermal cycling leading to thermal fatigue and its associated reliability problems.

This difference in thermal behavior of the two different applications can be explained as follows. The thread workloads of the *face recognition* application are characterized by longer duration of thread-independent high activity cycles followed by shorter duration of inter-thread dependent low activity cycles. When these threads are allocated to cores, the longer independent high-activity cycles of a thread overlap partially with the shorter dependent low-activity cycles of other threads. This is due to the Linux's default thread allocation, where threads are often migrated to balance load on the architecture. This leads to a higher temperature with lower thermal cycling. For the *mpeg2 encoding* application, the thread-independent high-activity cycles are shorter in duration, while the inter-thread dependent cycles are relatively long compared to the *face recognition* application. When these threads are allocated by Linux, only few of the available cores are being used. The default allocation results in a combination of independent high-activity cycles (of more than one threads), which overlap with each other (leading to a higher temperature) and similarly, the longer low-activity inter-thread dependent cycles overlap (leading to a lower temperature). This results in alternating high and low temperature triggering high thermal cycling.

Next, the same experiment is repeated by arbitrarily fixing the assignment of threads to cores (two cores execute two threads each and the other two cores execute one thread each) and leaving only the thread scheduling decision to the operating system. This is performed by changing all thread's affinity masks, forcing the Linux kernel to migrate these threads to the cores specified. The new thermal profiles are shown in blue in the same figure. As can be seen

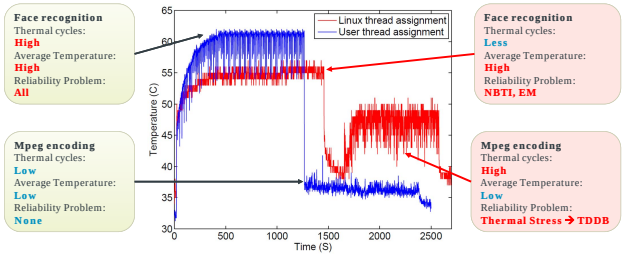

Figure 1. Thread-to-core affinity influences thermal profile

from the plot, this arbitrary thread assignment results in higher average temperature with higher thermal cycling for the *face recognition* application triggering all temperature related reliability concerns. This is because when threads are fixed to cores, the longer high-activity cycles of different threads overlap and so do the shorter low-activity cycles resulting in higher temperature and higher thermal cycling. When the same control is applied for *mpeg2 encoding*, the shorter high-activity cycles are not combined but are overlapped with each other. The temperature however increases, but for a shorter duration. This results in reduction of both the average temperature and thermal cycling thereby improving the lifetime. This example demonstrates two key aspects – thermal profile varies with application; and thread allocation influences thermal profile. This motivates the importance of an adaptive algorithm to learn the thermal behavior of an application and control it using appropriate thread-to-core assignment.

4. RELIABILITY COMPUTATION

4.1 Temperature Related MTTF

The lifetime reliability of a core is given by $R(t) = e^{-(t \cdot A)^{\beta}}$, where A is the *Thermal Aging* of the core (refer to [4, 15]).

$$A = \sum_{i} \frac{\Delta t_i}{t_p \times \alpha(T_i)} \tag{1}$$

where t_p is the execution time of the application, $\alpha(T_i)$ is the fault density (typically Weibull or Lognormal distribution) and T_i is the average temperature in the interval Δt_i. This equation allows to model any wear-out effect such as electromigration and negative bias temperature instability considered individually or as sum-of-failure-rate (SOFR). Mathematically, the MTTF of the core is

$$MTTF = \int_0^{\infty} R(t)dt = \int_0^{\infty} e^{-(t \cdot A)^{\beta}} dt \tag{2}$$

Thus, maximizing the MTTF is equivalent to minimizing the *aging* of the cores.

4.2 Thermal Cycling Related MTTF

Thermal cycling related MTTF is computed in three steps.

1. Calculating the thermal cycles from a thermal profile using Downing simple rainbow counting algorithm [5].

2. Calculating, from each thermal cycle, the number of cycles to failure using Coffin-Mansion's rule.

$$N_{TC}(i) = A_{TC} \left(\delta T_i - T_{Th}\right)^{-b} e^{\frac{E_a}{KT_{max}(i)}} \tag{3}$$

where $N_{TC}(i)$ is the number of cycles to failure due to i^{th} thermal cycle, A_{TC} is an empirically determined constant, δT_i is the amplitude of the i^{th} thermal cycle, T_{Th} is the temperature at which elastic deformation begins, b is the Coffin-Manson exponent constant, E_a is the activation energy and $T_{max}(i)$ is the maximum temperature in the i^{th} thermal cycle.

3. Calculating the MTTF using Miner's rule.

$$MTTF = \frac{N_{TC} \sum_{i=1}^{m} t_i}{m} \tag{4}$$

where t_i is the time for the i^{th} thermal cycle, m is the number of thermal cycles obtained in step 1 and N_{TC} is the effective cycles to failure determined using

$$N_{TC} = \frac{m}{\sum_{i=1}^{m} \frac{1}{N_{TC}(i)}} \tag{5}$$

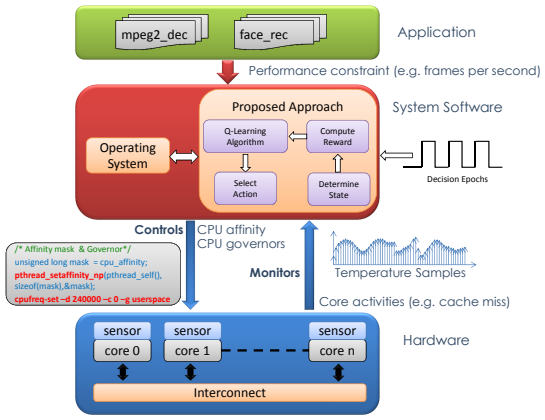

Figure 2. Proposed system level approach

Combining Equations 3-5, $MTTF = \frac{A_{TC}\sum_{i=1}^{m} t_i}{Thermal\ Stress}$, where *Thermal Stress* is an indication of the stress experienced by a core due to the thermal cycling. This is obtained using the following equation.

$$Thermal\ Stress = \sum_{i=1}^{m} (\delta T_i - T_{Th})^b \times e^{\frac{-E_a}{KTmax(i)}} \quad (6)$$

Thus, maximizing the MTTF of a core due to thermal cycling is equivalent to minimizing its *stress*.

5. PROPOSED APPROACH

Figure 2 illustrates the proposed dynamic thermal management approach as part of the system software layer and its interaction with the operating system, the application and the hardware layer. We adopted Q-Learning [18] as the reinforcement learning algorithm, where the learning agent maintains a Q-Table with entries corresponding to every state-action pair. The approach interfaces with the application layer to determine the performance requirement. For video applications such as *mpeg2 encoding/decoding*, the performance is measured as *frames per second* while for other applications such as *image processing*, performance refers to the execution time. A point to note here is that, although the above metrics are used for performance, the proposed algorithm can be trivially used with any reward function such as the performance counter based metric of [7].

The operating system samples the on board thermal sensors and reads the performance counters at the sampling interval to compute the thermal stress, aging and to monitor performance. The *stress* and *aging* values form the states of the Q-Table. The learning agent selects action at a fixed interval, called the "decision epoch". Each action comprises of thread affinity and voltage and frequency of operation and is enforced by the operating system on the underlying cores of the hardware layer. In most existing works on Q-learning based thermal management, the decision epoch is used to sample the temperature; actions are selected based on the instantaneous temperature from the sensor, which is not a true indication of the average temperature or thermal cycling in the interval. Since temperature-related reliability is governed by average temperature and thermal cycling, which need to be measured over a period of time, we used a sampling interval different than that of the decision epoch.

At the start of every decision epoch, the learning agent first computes the reward (or penalty) of the previous action based on three parameters – previous state, previous action and current state. The Q-Table entry corresponding to the previous state and previous action is updated according to the following equation [7, 18].

$$Q(\mathcal{E}_i, \aleph_i) \mathrel{+}= \alpha \left[R(\mathcal{E}_i, \mathcal{E}_{i+1}) + \gamma \max_{\aleph_j} \left(Q(\mathcal{E}_{i+1}, \aleph_j) - Q(\mathcal{E}_i, \aleph_i) \right) \right] \quad (7)$$

where α is the learning rate and is an indication of adaptation rate of the Q values and γ ($[0, 1]$) is the discount rate.

Subsequently, the learning agent selects the action with the highest Q-value for the current state. This action is first decoded by the operating system to determine the affinity masks for the threads and voltage and frequency of the cores. The threads are then migrated to the cores specified by the corresponding mask and the CPU governor settings are updated to enforce the voltage-frequency pair. Initially, the actions selected by the agent may not be optimal; the decisions are, however, refined over time. Algorithm 1 provides the pseudo-code of the learning agent. Details of this algorithm are provided in the following subsections.

ALGORITHM 1: Reinforcement Learning Algorithm

Input: Temperature reading from sensors T.
Output: Thread affinity and CPU governors $\aleph_i$.
Initialize the Q-Table, $\mathcal{E}_0 = 0$, $\aleph_0 = 0$, $i = 1$, $\alpha = 1$;
$TRec.push(T)$; //Record temperature;
if $|TRec| == Decision\ Epoch$ **then**
  Calculate $MA_s(i)$ and $MA_a(i)$;
  $\Delta MA_s = |MA_s(i) - MA_s(i-1)|$ and $\Delta MA_a = |MA_a(i) - MA_a(i-1)|$;
  if $((\Delta MA_s^L \leq \Delta MA_s < \Delta MA_s^U) \,||\, (\Delta MA_a^L \leq \Delta MA_a < \Delta MA_a^U))$ **then**
    $Q \leftarrow Q_{exp}$ and $\alpha \leftarrow \alpha_{exp}$; //Intra workload variation;
  end
  if $((\Delta MA_s \geq \Delta MA_s^U) \,||\, (\Delta MA_a \geq \Delta MA_a^U))$ **then**
    $Q \leftarrow Q_0$ and $\alpha \leftarrow 1$; //Inter workload variation;
  end
  $\mathcal{E}_i = IdentifyState(TRec)$;
  $R = CalculateReward(\mathcal{E}_{i-1}, \mathcal{E}_i)$ //Equation 8;
  $Q(\mathcal{E}_{i-1}, \aleph_{i-1}) = UpdateQtable(Q, R, \alpha)$ //Equation 7;
  $\aleph_i = SelectAction(Q, \mathcal{E}_i)$;
  $\alpha = UpdateLearningRate(\alpha, \mathcal{E}_i)$;
  $TRec = \emptyset$, $i = i + 1$;
end
sleep(temperature sampling interval);

5.1 Selecting the State and Action Space

The state space of our *Q-Learning* algorithm is composed of *stress* (determined using Equation 6) and *aging* (determined using Equation 1). To limit state space explosion, the working range of these parameters are divided into N_a and N_s disjoint intervals respectively. Specifically, *stress* is the set $\mathcal{S} = \{(0, s_0], (s_0, s_1], \cdots, (s_{N_s-1}, s_{N_s}]\}$ and the symbol $\hat{\mathfrak{s}}_i$ is used to represent the interval $(s_i, s_{i+1}]$. Similarly, *aging* is the set $\mathcal{A} = \{(0, a_0], (a_0, a_1], \cdots, (a_{N_a-1}, a_{N_a}]\}$ and the symbol $\hat{\mathfrak{a}}_i$ is used to represent the interval $(a_i, a_{i+1}]$. The environment is represented as $\mathcal{E} : (\mathcal{A} \times \mathcal{S})$. The action space of the agent is composed of thread affinity-based assignments and five CPU governors (*ondemand, conservative, performance, powersave* and *userspace*). The number of different affinity masks grows exponentially with the number of threads and cores. To restrict the action space, only a few of the alternatives are explored. Similarly, three frequency levels are selected for the *userspace* CPU governor. The action space is denoted by $\aleph : (\mathcal{M} \times \mathcal{G})$ where $\mathcal{M}$ is the set of thread affinity mappings and $\mathcal{G}$ is the set of governors.

5.2 Computing the Reward

The reward function $R(\mathcal{E}_i, \mathcal{E}_{i+1})$ is given by

$$R(\mathcal{E}_i, \mathcal{E}_{i+1}) = \begin{cases} -\hat{\mathfrak{s}}_i \times \hat{\mathfrak{a}}_i & \text{if } (\hat{\mathfrak{s}}_i = \hat{\mathfrak{s}}_{N_s}) \text{ or } (\hat{\mathfrak{a}}_i = \hat{\mathfrak{a}}_{N_a}) \\ f(\hat{\mathfrak{a}}_i, \hat{\mathfrak{s}}_i) + (P_c - P) & \text{otherwise} \end{cases} \quad (8)$$

where P is the performance, P_c is the performance constraint and the function f is determined empirically as $f = (a.K_1.stress + b.K_2.aging)$, where a and b are relative importance of *stress* and *aging*: For mpeg (large thermal cycles), $a > b$ and for tachyon (high average temperature), $b > a$. Two sets of a and b values are used based on the mean of *stress* and *aging*. $K_1(K_2)$ is the learning weight and is a Gaussian function of the *stress* (*aging*) values. This distribution assigns lower rewards to thermally unstable as well as the thermal stable states and thus allows the algorithm to explore other states and prevent Q-Table clustering.

For the design of the reward function, two cases are considered. If the *stress* or *aging* falls in the unsafe zone (the last interval), the decision is penalized. This is indicated with a negative value of the reward function, which decreases the Q value (refer to Equation 7) so that the corresponding action is avoided in the future. For all other cases, the reward function is composed of performance penalty and the thermal safety of the state. Specifically, if the performance requirement is not satisfied, $(P_c - P)$ is negative and the reward (or penalty) is governed by the function f. Finally, rewards are guaranteed if an action leads to a thermal safe state while satisfying the performance requirements.

5.3 Learning Phases

The Q-learning algorithm is composed of three phases – *exploration*, *exploration-exploitation* and *exploitation*. In the *exploration* phase, the agent selects action arbitrarily to determine the corresponding reward. This phase is characterized by α values close to 1 to enable a significant fraction of the reward values to contribute towards the Q-Table entries. In the *exploration-explotation* phase, the agent selects the best action (with the highest Q value) for the current state and updates the Q-Table entry with a part of the reward value. Finally, in the *exploitation* phase, the algorithm still selects the action corresponding to the highest

Table 2. MTTF (in years) of reinforcement learning algorithm for three applications. The scaling parameters for computing MTTF are so selected such that the MTTF of an unstressed core (i.e. an idle core) is 10 years.

Benchmarks		Average Temperature (^oC)			Peak Temperature (^oC)			Thermal cycling MTTF			Average temperature MTTF		
Application	Data	Linux	Ge et. al [7]	Proposed	Linux	Ge et. al [7]	Proposed	Linux	Ge et. al [7]	Proposed	Linux	Ge et. al [7]	Proposed
tachyon	set 1	69.2	52.6	50.6	71.5	63.0	60.0	7.1	2.3	5.5	0.7	3.0	3.6
	set 2	50.5	44.5	43.8	57.3	56.3	52.0	2.8	4.3	5.3	2.6	4.5	4.8
	set 3	50.8	44.7	41.6	57.8	54.5	48.8	1.3	3.8	6.5	2.4	4.1	5.5
mpeg dec	clip 1	36.0	34.0	34.2	42.7	41.3	39.0	2.1	0.8	6.4	3.7	4.5	4.4
	clip 2	35.6	34.4	34.2	42.3	42.0	39.3	1.1	0.9	4.7	3.8	4.3	4.4
	clip 3	34.3	34.4	34.0	43.0	39.7	44.3	1.6	3.4	3.7	4.3	4.2	4.5
mpeg enc	seq 1	33.7	34.1	32.6	41.0	40.7	40.3	4.3	4.4	5.2	4.6	4.5	5.2
	seq 2	34.4	33.5	32.3	41.3	39.7	41.7	3.9	6.2	4.8	4.3	4.7	5.4
	seq 3	33.2	33.7	31.8	40.3	40.0	41.0	4.6	5.1	5.1	4.9	4.6	5.7

Q-value for the current state; however, the Q-table entries are not updated (or updated with negligible fraction of the reward value). This phase is characterized with α values close to 0. To facilitate transition between the three phases of the algorithm, an exponentially decreasing function is selected for the α value. This function is implemented in the $UpdateLearningRate$ subroutine.

5.4 Adaptation to Workload Variation

To incorporate intra- and inter-application workload variations, moving averages of the stress and the aging are determined at the start of every decision epoch. The change in the moving averages are identified in the algorithm as ΔMA_s and ΔMA_a. Two thresholds are maintained for each of these quantities identified with the superscript L and U, respectively. The learning agent considers a change in the moving average as intra-application variation if the change is greater than the lower threshold and lower than the upper threshold (for example when $\Delta MA_s^L \leq \Delta MA_s < \Delta MA_s^U$). This causes the Q-table to be updated with the Q values from the end of the exploration phase. The alpha value is also updated accordingly. On the other hand, the agent considers a change in the moving average as inter-application variation, if the change is greater than the upper threshold. In this case, the Q-Table entries are initialized to 0 and α value initialized to 1 to start learning again. This is the implicit learning technique adopted in this work. The α_{exp}, lower and upper threshold are determined empirically from the set of applications at hand. A point to note here is that, the agent maintains two Q-Tables – one with static Q values from the end of the exploration phase and the other with Q values that are updated at each decision epoch.

Algorithm 1 provides the pseudo-code of the reinforcement learning algorithm. Important parameters of the algorithm are the temperature sampling interval, decision epoch and the number of states and actions. We adopt a systematic approach to decide these parameters.

6. RESULTS AND DISCUSSIONS

The proposed run-time approach is validated experimentally on an Intel quad-core CPU running Linux kernel 3.8.0. Performance is monitored using *perf* [1]; temperature is measured by sampling the thermal sensors directly; power/energy consumption is recorded using *likwid powermeter* [16]. A set of multi-threaded multimedia applications are considered from the *ALPBench* [11] benchmark suite. These benchmarks are *mpeg enc, mpeg dec, face recognition, sphinx* and *tachyon* and are representative of the multimedia workloads for most multicore systems. The number of threads in each of these applications is configurable and, in this paper, six threads are considered to exploit the full benefit of the four cores. The device parameters used for computing the *aging* and *stress* of a core are the same as that used in [2, 17].

6.1 Intra-Application

Table 2 reports the average temperature, peak temperature and MTTF due to average temperature (equivalently *aging*) and thermal cycling (equivalently *stress*) of the proposed technique in comparison with Linux's *ondemand* [13] and the technique proposed in [7]. Results are reported for three different applications, each of which are executed for three sets of input data. There are a few trends to follow from this table. First, the technique in [7] minimizes instantaneous temperature achieving a lower *aging* (higher MTTF) than Linux (refer to columns 3-4 & columns 12-13). This signifies the importance of the thermal management feature for operating systems. However, thermal cycling is not accounted for in this technique and therefore does not guarantee reduction of *stress*. This is evident from the thermal cycling-related MTTF values for scenarios such as *tachyon* on set 1 and *mpeg dec* on clip 1, where the MTTF values obtained using [7] is lower than that of Linux.

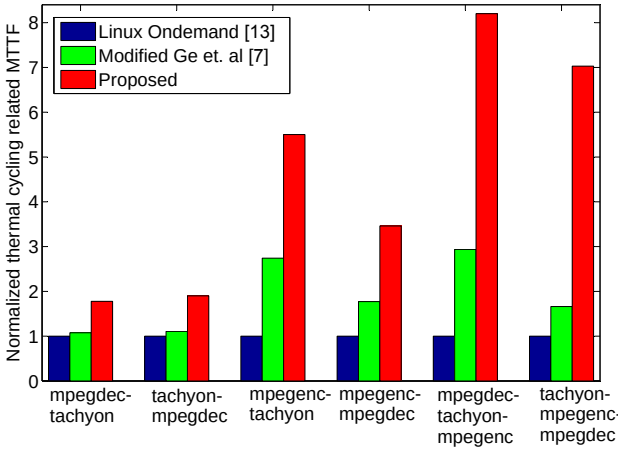

Figure 3. Inter-application results

Second, the proposed reinforcement learning algorithm minimizes the average temperature by upto 18.6^oC and the peak temperature by upto 11.5^oC. This reduction leads to an improvement in MTTF due to *aging* by upto 5x (average 82%). For applications such as *mpeg dec*, the improvement is less as the average temperature is usually lower, leading to a limited scope to further improve *aging*. The thermal cycling effect dominates in this application. For other applications such as *tachyon*, the improvement is significant due to the large scope for improving both *aging* and *stress*. Third, the proposed adaptive algorithm also minimizes thermal cycling which is not considered in Linux and [7]. This is highlighted in columns 9-11. An important point to note is that for the *tachyon* application with set 1 data, the MTTF due to *stress* using Linux's default thread assignment is higher (7.1 years); however, the MTTF due to *aging* is lower (0.7 years). The proposed technique balances the two effects and improves the MTTF due to *aging* by 5x with less than 25% sacrifice in MTTF due to *stress* (while still maintaining a satisfactory MTTF of 5.5 years). For all other applications and data sets, the proposed reinforcement learning algorithm outperforms Linux in terms of thermal cycling by an average 2.3x.

Last, the proposed approach outperforms [7] both in terms of *aging* (an average 13% higher average temperature related MTTF) and *stress* (an average 2x higher thermal cycling related MTTF). While the improvement of thermal cycling is expected (as this is incorporated explicitly in the proposed approach), the improvement in *aging* is due to the following. First, the decoupling of the temperature sampling interval from the decision epoch (enabling a finer control on the average temperature); second, careful choice of the design parameters as demonstrated in Figure 6 & 7.

6.2 Inter-Application

Figure 3 plots the normalized MTTF due to thermal cycling obtained using the proposed technique in comparison with that obtained using the modified technique of [7] for six different inter-application scenarios. The MTTF values are normalized with respect to the MTTF obtained using Linux's *ondemand* governor. Furthermore, the technique of [7] is modified to consider application switching using explicit indication from the application layer. The proposed approach however, detects application switching autonomously (without communication from the application layer) and performs re-learning (as discussed in Section 5).

There are six inter-application scenarios considered in this experiment. A scenario *appA-appB* indicate that *appA* is

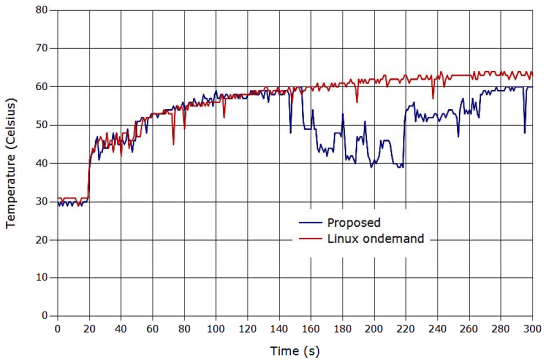

Figure 4. Exploration phase of the learning algorithm

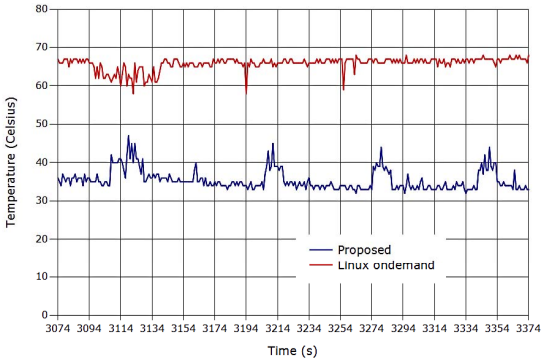

Figure 5. Exploitation phase of the learning algorithm

executed first followed by application *appB*. As can be seen from the figure, the technique of *Ge et. al* [7] results in higher MTTF than Linux. For some inter-application scenarios such as *mpegdec-tachyon* and *tachyon-mpegdec*, the improvements are less ($\approx$ 8%). For other inter-application scenarios, this improvement is higher. On average, for all the scenarios, [7] increases MTTF by 80% as compared to Linux. The approach proposed in this paper outperforms both Linux and that of [7] in terms of thermal cycling, achieving 5x improvement with respect to Linux and 3x improvement with respect to [7]. A point to note from the figure is that, the MTTF improvement (over [7]) using the proposed approach, increases with frequent application switching. This is evident from the higher MTTF improvement of 3.5x obtained for the three-application scenarios (*mpegdec-tachyon-mpegenc* and *tachyon-mpegenc-mpegdec*) as compared to improvements obtained for the four other two-application scenarios. Thus, the proposed approach improves thermal cycling related MTTF significantly for inter-application scenarios. The improvement increases with an increase in application switching (typical of a modern multicore system).

6.3 Phases of the Reinforcement Learning Algorithm

To further demonstrate the temperature profile obtained using the proposed algorithm, Figures 4 and 5 plot the exploration and the exploitation phases, respectively in comparison with Linux's popular and default *ondemand* governor for the *face recognition* application. As can be seen, at the beginning of the exploration phase, the temperature obtained using the proposed algorithm is comparable to that obtained using Linux. This is because, at this phase, the proposed algorithm explores the impact of CPU affinity and operating frequency choices on the temperature of a core. However, when the proposed model learns the impact of these parameters on temperature (i.e. in the exploitation phase), the CPU affinities and operating frequencies are selected such that the average temperature is reduced (Figure 5).

6.4 Selection of Design Parameters

Figure 6 plots the impact of varying the interval of sampling temperature from the on-board thermal sensors on *thermal stress* and performance for the *tachyon* application. The figure also plots the auto-correlation of temperature samples which is a measure of how much the temperature values change across consecutive samples. Specifically, a high auto-correlation indicates a small difference between consecutive thermal data. As can be seen from the figure, the

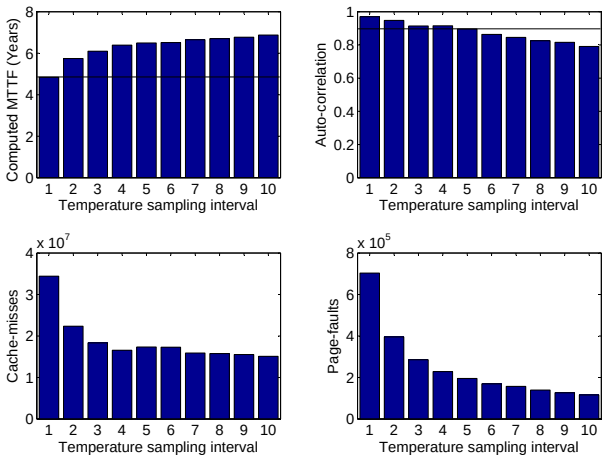

Figure 6. Impact of temperature sampling interval (sec)

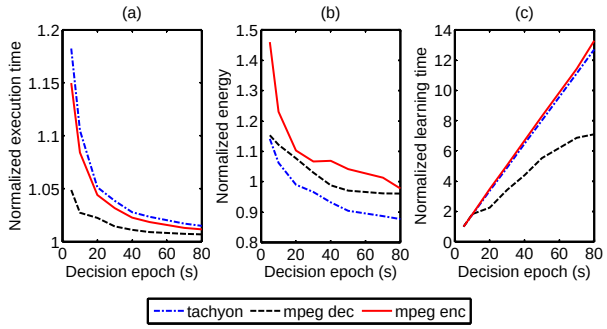

Figure 7. Effect of the length of decision epoch

auto-correlation is high for lower sampling intervals. This is expected as temperature variation is usually slower, being dependent on the thermal property of silicon, ambient temperature and cooling technology used. The MTTF value increases with an increase in size of the sampling interval. This is, however, an over estimation of the actual MTTF value (corresponding to the sampling interval of 1 sec). This overestimation is due to the loss of temperature accuracy with increasing sampling interval resulting in a lower *stress* and hence higher MTTF. Finally, the number of cache-misses and page faults decrease with an increase in the sampling interval, clearly signifying the performance improvement. Based on these trade-offs, a sampling interval of 3 sec is selected for the *tachyon* application. Similarly, the sampling interval for other applications are determined. An interval of 3 sec provides the best trade-off for most of the applications. In future, determination of the sampling interval can be incorporated as part of the learning algorithm itself.

Figure 7 plots the performance of the algorithm with increasing decision epochs for the three applications. The execution time, dynamic energy consumption and the adaptation time are compared. The execution time refers to the completion time with a fixed length of input data. For the *mpeg enc* (or *mpeg dec*) application, this is the time to encode(decode) 10MB of video. For the *tachyon* application, the time is measured as the rendering time of 300 images. The execution time using the reinforcement learning algorithm is normalized with respect to the execution time of Linux (with no adaptation) for the same input data. As can be seen from Figure 7(a), for all applications, the execution time overhead is higher for smaller decision epochs due to the overhead associated with frequent decision changes. The execution time overhead reduces with larger decision epochs. Figure 7(b) reports the dynamic energy consumption of the proposed approach for the same input data normalized with respect to the Linux (without adaptation). As expected, the energy consumption is also higher for smaller decision epochs due to the frequent adaptation of the approach. Finally, Figure 7(c) plots the training time i.e. the time required for the algorithm to learn the thermal behavior of an application. The results are normalized with respect to the training time of the algorithm with a decision epoch of 5sec. As can be seen, the training time increases with an increase in the decision epoch. This is because the training time is a function of decision epoch and number of iterations. The decision epoch for our algorithm is selected based on this

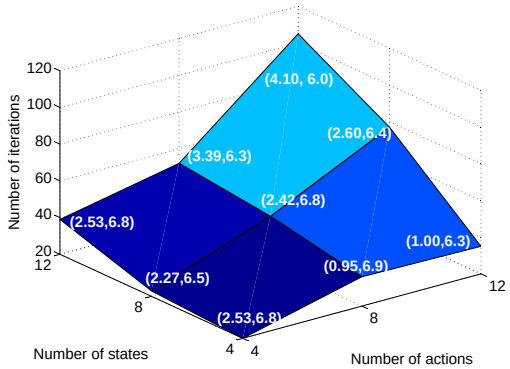

Figure 8. Convergence of the reinforcement learning algorithm

Table 3. Execution time (in sec) of the proposed approach.

App.	Linux				Ge et. al [7]	Proposed
	ondemand	powersave	2.4GHz	3.4GHz		
tachyon	629	1337	913	627	1137	810
mpeg dec	1208	1222	1183	1127	1328	1204
mpeg enc	1623	1655	1628	1571	1676	1599

energy, execution and training time trade-off.

Figure 8 plots the convergence time of the proposed algorithm for the *mpeg decoding* application with a varying number of states and actions. The convergence time is measured by the number of decision epochs needed to train the proposed learning algorithm. As can be seen from the figure, the number iterations i.e. the training time increases with an increase in the number of actions and/or states. This is expected as an increase in the states and/or actions leads to an increase in the size of the Q-Table and therefore more iterations are needed to learn (fill the table entries). The figure also reports the MTTF as coordinates (*stress*, *aging*) for each design point. As the size of the Q-Table increases, the reinforcement learning algorithm has finer control on the temperature resulting in an improvement in the MTTF. The number of states and actions are chosen based on this learning time and solution quality trade-off.

6.5 Performance and Energy Trade-offs

Table 3 reports the execution time of the proposed approach in comparison with the one proposed in [7] and the Linux-based approach for *ondemand*, *powersave* and *user-space* power governors. Two user frequencies (2.4GHz and 3.4GHz) are shown in the table. The execution time with the highest frequency of 3.4GHz is the least for all the applications. This is expected as the execution time decreases with an increase in frequency. For the same reason, the execution time with the lowest frequency (*powersave*) is the highest. However, the power overhead is least (as explained next). For some applications such as *tachyon*, the proposed approach has higher execution time than the Linux's *ondemand* governor by upto 30%. This is because the workload in the *tachyon* application forces the kernel to execute always at the highest frequency of 3.4GHz in the *ondemand* power mode. Thus, the execution time for Linux's *ondemand* and *userspace-3.4GHz* are comparable. The proposed approach on the other hand explores different power modes to reduce thermal *stress* and *aging* and therefore trades-off performance. For other applications such as *mpeg enc* and *mpeg dec*, the execution time of the proposed approach is lower than that of Linux's *ondemand* power mode. Finally, with respect to [7], the proposed approach reduces execution time by an average 14%.

Figure 9 plots the average dynamic power and energy consumption (measured using *likwid-powermeter*) of the proposed algorithm in comparison with that of [7] and the Linux governors. Although the power and energy overhead are not reported in [7], these are calculated here for a comparative study. As can be seen from the figure, the proposed approach reduces power consumption by an average 6% in comparison with Linux *ondemand* governor with 10% increase in execution time. Although the dynamic power consumption of [7] is lower than the proposed approach (on average 4% lower), the energy consumption (which incorporates both power and performance) of the proposed approach is 10% lower than [7] (within 3% of the energy consumption of Linux's *ondemand* governor).

An interesting point to note here is that, by reducing the average temperature the proposed technique improves the

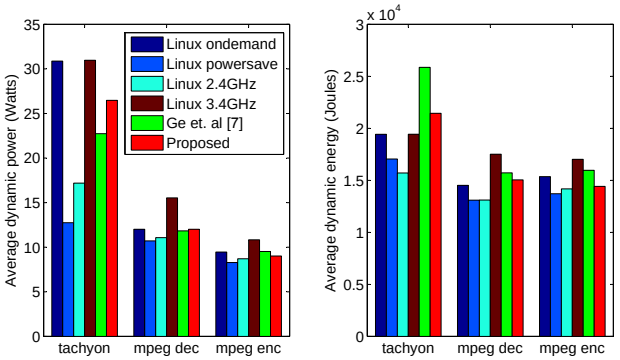

Figure 9. Power comparison of the reinforcement learning algorithm

leakage power. Although leakage power is not measured on the board, the established models for the same (such as the one proposed in [17]) indicate that the proposed algorithm improves leakage energy by an average 15% as compared to Linux's *ondemand* governor and 11% as compared to [7].

7. CONCLUSIONS

We have proposed a reinforcement learning-based run-time approach for multicore system to adapt to thermal variations both within an application as well as when the system switches from one application to another. The control is provided by overriding the operating system mapping decisions using affinity masks and dynamically changing the frequency of cores using CPU governors. The approach is validated experimentally using an Intel quad-core platform running Linux kernel 3.8.0. Results demonstrate that the proposed approach is able to improve MTTF by an average 2x for intra-application and 3x for inter-application scenarios as compared to the existing dynamic thermal management technique. Furthermore, the approach also improves dynamic energy consumption by an average 10% and static energy by 11%. In future, the approach can be extended to consider concurrent applications and heterogeneous cores.

8. ACKNOWLEDGMENTS

This work was supported by the Engineering and Physical Sciences Research Council Programme Grant, EP/K034448/1. See www.prime-project.org for more information about the PRiME programme.

References

[1] Perf: Linux Profiling with Performance Counters, https://perf.wiki.kernel.org. 2012.
[2] T. Chantem et al. Enhancing Multicore Reliability Through Wear Compensation in Online Assignment and Scheduling. In *DATE*, 2013.
[3] A. K. Coskun, T. S. Rosing, et al. Temperature management in multiprocessor socs using online learning. In *DAC*, 2008.
[4] A. Das et al. Temperature Aware Energy-Reliability Trade-offs for Mapping of Throughput-Constrained Applications on Multimedia MPSoCs. In *DATE*, 2014.
[5] S. Downing and D. Socie. Simple Rainflow Counting Algorithms. *International Journal of Fatigue*, 1982.
[6] T. Ebi et al. Economic Learning for Thermal-aware Power Budgeting in Many-core Architectures. In *CODES+ISSS*, 2011.
[7] Y. Ge and Q. Qiu. Dynamic Thermal Management for Multimedia Applications Using Machine Learning. In *DAC*, 2011.
[8] V. Hanumaiah and S. Vrudhula. Temperature-Aware DVFS for Hard Real-Time Applications on Multicore Processors. *IEEE Transactions on Computers*, 2012.
[9] R. Jayaseelan and T. Mitra. Dynamic Thermal Management via Architectural Adaptation. In *DAC*, 2009.
[10] W. Lee et al. GOP-level Dynamic Thermal Management in MPEG-2 Decoding. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2008.
[11] M.-L. Li et al. The ALPBench Benchmark Suite for Complex Multimedia Applications. In *Workload Characterization Symposium*, 2005.
[12] F. Mulas et al. Thermal Balancing Policy for Multiprocessor Stream Computing Platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2009.
[13] V. Pallipadi and A. Starikovskiy. The Ondemand Governor. In *Proceedings of the Linux Symposium*, 2006.
[14] K. Skadron et al. Temperature-Aware Microarchitecture: Modeling and Implementation. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2004.
[15] J. Srinivasan et al. The Case for Lifetime Reliability-Aware Microprocessors. In *International Symposium on Computer Architecture (ISCA)*, 2004.
[16] J. Treibig et al. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. In *International Conference on Parallel Processing Workshops (ICPPW)*, 2010.
[17] I. Ukhov et al. Steady-state Dynamic Temperature Analysis and Reliability Optimization for Embedded Multiprocessor Systems. In *DAC*, 2012.
[18] C. J. Watkins et al. Q-Learning. *Machine learning*, 1992.