Workload Change Point Detection for Run-time Thermal Management of Embedded Systems

Anup Das, *Member, IEEE,* Geoff V. Merrett, *Member, IEEE,*
Mirco Tribastone, *Member, IEEE,* and Bashir M. Al-Hashimi *Fellow, IEEE,*

***Abstract*—Applications executed on multicore embedded systems interact with system software (such as the OS) and hardware, leading to widely varying thermal profiles which accelerate some aging mechanisms, reducing the lifetime reliability. Effectively managing the temperature therefore requires (1) autonomous detection of changes in application workload and (2) appropriate selection of control levers to manage thermal profiles of these workloads. In this paper we propose a technique for workload change detection using density ratio-based statistical divergence between overlapping sliding windows of CPU performance statistics. This is integrated in a run-time approach for thermal management, which uses reinforcement learning to select workload-specific thermal control levers by sampling on-board thermal sensors. Identified control levers override the OS's native thread allocation decision and scale hardware voltage-frequency to improve average temperature, peak temperature and thermal cycling. The proposed approach is validated through its implementation as a hierarchical run-time manager for Linux, with heuristic-based thread affinity selected from the upper hierarchy to reduce thermal cycling and learning-based voltage-frequency selected from the lower hierarchy to reduce average and peak temperatures. Experiments conducted with mobile, embedded and high performance applications on ARM-based embedded systems demonstrate that the proposed approach increases workload change detection accuracy by an average 3.4x, reducing the average temperature by 4–25°C, peak temperature by 6–24°C and thermal cycling by 7–35% over state-of-the-art approaches.**

***Index Terms*—Embedded System, run-time manager, thermal optimization, change point detection.**

I. Introduction

Modern mobile embedded systems execute applications that interact with system software (the OS) and hardware differently, generating widely varying thermal profiles and increasing the power consumption [1]. Earlier studies have shown improvement in average and peak temperatures by dynamically switching the voltage and frequency of processing cores (**hardware control** lever) [2]. An emerging concern for embedded systems is thermal cycling, i.e. the wear-out induced by thermal stress due to a mismatched coefficient of thermal expansion of adjacent material layers. A recent study [3] has shown that OS thread affinity (**software control** lever) has significant impact on thermal cycling. A combination of hardware and software control levers is effective in alleviating all three thermal concerns – peak temperature, average temperature and thermal cycling, leading to a reduced power consumption and an increase in lifetime. However, determining the optimum OS thread affinity to minimize thermal overhead while satisfying the performance constraint is an NP-hard problem; exploring the entire state space is usually difficult at run-time. The technique of [3] explores a limited subset of this state-space determined empirically at design-time for typical application workloads.

On the other side, an application's thermal behavior changes within execution duration, requiring different combinations of thermal control levers. Existing techniques cannot detect application changes at run-time, and therefore a single choice is made at the start of application execution. We demonstrate in Section V, an incorrect choice of control levers can negatively impact the performance and results in higher thermal and hence, power overheads.

In this paper we address the following challenges for run-time thermal optimization of embedded systems.

- detecting application workload changes at run-time;
- determining the right combination of hardware and software control levers once a change is detected; and
- solving the NP-hard thread affinity problem at run-time.

Contributions: Our contributions are as follows:

- mathematical formulation of the thermal optimization problem for embedded systems (Section II);
- application change detection using density ratio-based statistical divergence between overlapping sliding windows of CPU performance statistics (Section III);
- a reinforcement learning-based low overhead run-time approach, realized as a run-time manager for Linux (Section IV);
- a two-step hierarchical approach for thread affinity and voltage-frequency selection, enabling finer control on temperature and addressing scalability (Section IV); and
- validation with mobile, embedded and CPU-intensive applications on different ARM-based platforms (Section V).

This work extends our earlier work [3] by (1) run-time application change detection and (2) two-stage hierarchical approach for selecting thread affinity and hardware voltage-frequency. Additionally, we provide a low-overhead run-time manager implementation of the approach for Linux OS validated across different ARM-based embedded systems.

Experiments conducted with embedded and CPU-intensive applications from MiBench, PARSEC and the SPLASH2

A. Das, G. V. Merrett, M. Tribastone and B. M. Al-Hashimi are with the Department of Electronics and Computer Science, University of Southampton, Southampton, UK, SO17 1BJ. E-mail: {a.k.das,gvm,bmah}@ecs.soton.ac.uk

Manuscript received April 21, 2015; revised September 08, 2015.

TABLE I
MTTF CONSIDERING DIFFERENT WEAR-OUT MECHANISMS [4] [5].

Wear-out	MTTF	Comments
Electromigration (EM)	$\frac{A_{EM}}{J^n}\exp\left(\frac{E_{a_{EM}}}{KT}\right)$	A_{EM} is a material-dependent constant, J is the current density, n is empirically determined constant with a typical value of 2 for stress related failures, $E_{a_{EM}}$ is the activation energy of electromigration, K is the Boltzman's constant, and T is the temperature.
Negative Bias Temperature Instability (NBTI)	$\frac{A_{NBTI}}{(V_{GS})^\gamma}\exp\left(\frac{E_{a_{NBTI}}}{KT}\right)$	A_{NBTI} is a constant dependent on the fabrication process, γ is the voltage acceleration factor and $E_{a_{NBTI}}$ is the activation energy.
Time Dependent Dielectric Breakdown (TDDB)	$A_{TDDB}\cdot A_G\cdot\left(\frac{1}{V_{GS}}\right)^{\alpha-\beta T}\exp\left(\frac{X}{T}+\frac{Y}{T^2}\right)$	V_{GS} is the gate voltage, T is the temperature, α, β, X and Y are fitting parameters, A_G is the surface area of the gate oxide and A_{TDDB} is an empirically determined constant.
Stress Migration (SM)	$A_{SM}\lvert T_0-T\rvert^{-n}\exp\left(\frac{E_{a_{SM}}}{KT}\right)$	A_{SM} is a material dependent constant, T_0 is the metal deposition temperature and $E_{a_{SM}}$ is the activation energy.
Thermal Cycling (TC)	$\frac{A_{TC}\sum_{i=1}^{m}t_i}{\textit{Thermal Stress}}$	*Thermal Stress* is an indication of the stress experienced due to the thermal cycling. This is obtained using *Thermal Stress* $=\sum_{i=1}^{m}(\delta T_i-T_{Th})^c\times e^{\frac{-E_a}{KT_{max}(i)}}$, 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, c is the Coffin-Manson exponent constant, E_a is the activation energy of thermal cycling and $T_{max}(i)$ is the maximum temperature in the i^{th} thermal cycle.

benchmarks suites, along with applications typically executed on modern smartphones demonstrate that the proposed learning-based run-time approach increases workload change detection accuracy by an average 3.4x, reducing average temperature by 4-25°C, peak temperature by 6-24°C and thermal cycling by 7-35% over state-of-the-art approaches.

II. Problem Formulation

The leakage power consumption of a system and its lifetime reliability are both dependent on temperature [4] [5]. In this section, we formulate the objective we optimize in this work.

A. Optimization Objective

As seen from Table I, three important parameters to optimize are average temperature, peak temperature and thermal cycling, which are combined into a single objective, **thermal overhead** (T_O). This is computed as follows. Let $T_1^i, T_2^i, \cdots, T_{N_l}^i$ are N_l thermal sensor readings collected at regular intervals in time duration t_i to t_{i+1}. The system thermal overhead in this interval is given by

$$T_O(t_i \rightarrow t_{i+1}) = \begin{cases} \text{mean}(T_1^i, \cdots, T_{N_l}^i) + \omega \cdot \max(T_1^i, \cdots, T_{N_l}^i) \\ \texttt{ThermalCycle}(T_1^i, \cdots, T_{N_l}^i) \end{cases} \quad (1)$$

where thermal overhead is computed as (1) the weighted sum of the mean and the max temperatures for thermal optimization; or (2) the thermal cycling using the function `ThermalCycle` for thermal cycling related reliability (see Appendix A for details) optimization. The weight ω is adjusted automatically at run-time depending on the thermal throttling limit, specified thermal safe value, and how frequently an application reaches the critical temperature.

B. Choice of Machine Learning

Temperature of an embedded system depends on

- application, e.g. type of instructions executed
- architecture, e.g. dynamic power consumed per instruction and memory access, leakage power, floorplan, presence of heat sink etc.
- environment, e.g. ambient temperature

These dependencies are are not provided for commercial off-the-shelf embedded systems. Common practice is to pre-characterize the platform [6] to model some of these dependencies. Additionally, hardware and software thermal control levers effect temperature differently, depending on application's workload, its cross-layer interactions with system software and hardware, and on the working environment. Therefore, we use machine learning to identify these interactions and to select appropriate control levers, to minimize the long-term thermal overhead of a system.

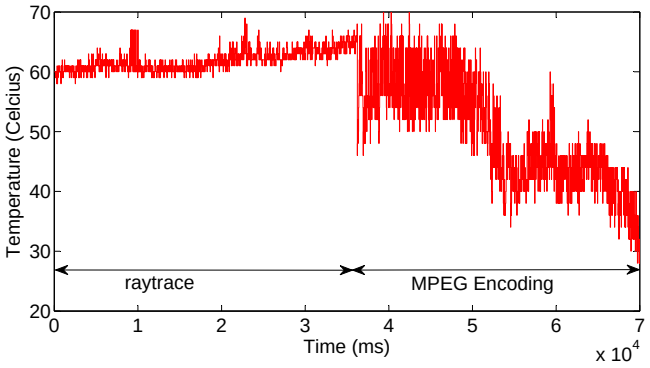

Fig. 1. Temperature collected during execution of raytrace followed by MPEG encoding.

C. Need for Application Change Detection

Workload on an embedded system changes both within execution of an application and also when the system switches application. Workload changes result in a different thermal profile as shown in Figure 1, which plots CPU temperature during execution of the raytrace and the MPEG encoding applications. As seen from this figure, the thermal profile

changes at 35s when there is a switch from raytrace to MPEG encoding. Furthermore, the average temperature changes at 52s during video encoding. This is due to a change in the frames per second (fps) requirement from 60 to 24 fps at this time. In comparison to the raytrace workload, the MPEG encoding workload results in lower average temperature and higher thermal cycling. This example illustrates that different thermal control levers are required for processing different workload segments, and thus necessitating an autonomous approach for detecting workload changes.

III. Application Change Detection

Modern processor cores (such as the ARM A-Series and the Intel IvyBridge) are equipped with performance management unit (PMU) consisting of registers and counters to record hardware events during application execution. An important event relevant to this work is the **CPU cycles**, defined as the number of clock cycles consumed to execute a workload (or a segment of it). Workload phase detection has recently been an active subject of research [7] [8]. These approaches are based on multi-variable offline characterization using principle component analysis. Contrary to these approaches, our work is a run-time only approach using CPU cycles. We have chosen a single variable to reduce complexity for real-time requirements and is shown to be effective (Section V). CPU cycles is specifically selected because it provides a reasonably accurate estimate of workload phase change.

CPU cycles collected at a regular interval during application execution form **time-series** data; Figure 2 plots CPU cycles for two applications – (a) the sobel filter implemented using openCV and (b) the fluidanimate. **Change points** in this workload (indicated by red dashed lines) are time instances where statistical characteristics such as mean, standard deviation etc. differ. Objective of this work is to detect these change points autonomously. This can be achieved by using the statistical distance between two sliding windows as shown in Figure 3. In this technique, $(2N-b)$ CPU cycles count are collected in two windows (termed as the test and the reference window, respectively) of size N each. After executing each workload segment, the CPU cycles count is pushed in the test window and all other CPU cycles count are right shifted (hence the name **sliding window**). Test and the reference window have b CPU cycles in common; this is defined as the **overlapping depth** of the technique. CPU cycles in the current and $(2N-b-1)$ previous workload segments constitute a total of $(2N-b)$ CPU cycles in these two windows. Total storage requirement of the sliding window-based detection technique is $(2N-b)*N_{Bits}$. The statistical distance between the test and the reference window is given by

$$d(W_{test}, W_{ref}) = D(W_{test} \mid\mid W_{ref}) + D(W_{ref} \mid\mid W_{test}) \quad (2)$$

where the divergence measure D is defined as

$$D(W_{test} \mid\mid W_{ref}) = \int g\left(\frac{P^{test}(x)}{P^{ref}(x)}\right) dx \quad (3)$$

where $P^{test}(x)$ and $P^{ref}(x)$ are the probability density functions (or simply the **densities**) of the CPU cycles in the test and the reference window, respectively; the function $g()$ defines the type of divergence – for Kullback-Leibler divergence, $g(t) = t\log t$ and for Pearson divergence, $g(t) = \frac{1}{2}(t-1)^2$. As shown in [9], estimating the density from time-series data is an NP-hard problem [10]. A common alternative is to estimate the **density ratio** instead [10] [11].

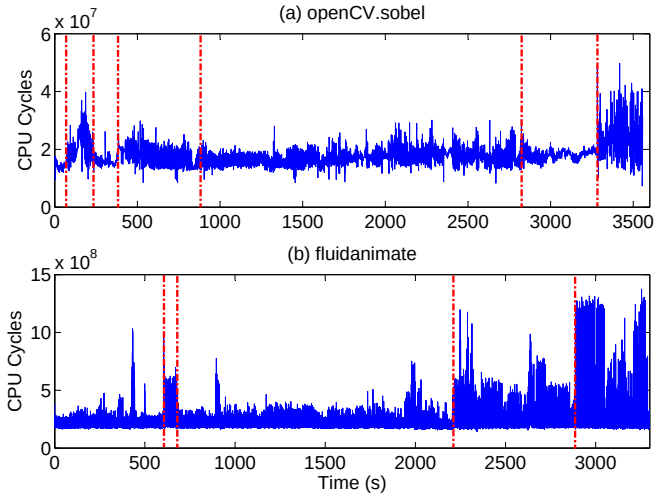

Fig. 2. CPU cycles collected during execution of (a) openCV.sobel and (b) fluidanimate. Changes in execution phase are marked by red lines.

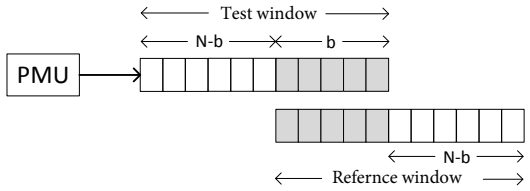

Fig. 3. Overlapping sliding window of CPU cycles.

Let the density ratio $\frac{P^{test}(x)}{P^{ref}(x)} = h(x)$ be estimated using a linear model as

$$\frac{P^{test}(x)}{P^{ref}(x)} = h(x) \approx \hat{h}(x;\mathbf{\Theta}) = \mathbf{\Theta}^T\mathbf{K}(x) = \frac{\hat{P}^{test}(x)}{P^{ref}(x)} \quad (4)$$

where $\mathbf{\Theta} = (\theta_1 \ \cdots \ \theta_N)^T$ is the parameter to be determined from the time-series data and $\mathbf{K}(x) = (\mathcal{K}_1(x) \ \cdots \ \mathcal{K}_N(x))^T$ is a Gaussian kernel basis function centered around the test window and is defined as

$$\mathcal{K}_j(x) = \exp\left(-\frac{\|x - W^{test}(j)\|^2}{2\sigma^2}\right) \quad (5)$$

The estimated density is $\hat{P}^{test}(x) = \hat{h}(x;\mathbf{\Theta})\,P^{ref}(x)$. The parameter $\mathbf{\Theta}$ is selected such that the estimate $\hat{h}(x;\mathbf{\Theta})\,P^{ref}(x)$ is as close as possible to the actual density $P^{test}(x)$. In other words, $\mathbf{\Theta}$ is selected to minimize the divergence between the actual density and its estimate. For demonstration, the Kullback-Leibler (KL) divergence between them is defined as

$$\int P^{test}(x)\log\frac{P^{test}(x)}{\hat{h}(x;\mathbf{\Theta})\,P^{ref}(x)}dx \quad (6)$$
$$= \int P^{test}(x)\log\frac{P^{test}(x)}{P^{ref}(x)}dx - \int P^{test}(x)\log\left(\hat{h}(x;\mathbf{\Theta})\right)dx$$

To minimize this KL divergence, it is essential to maximize the second term of Equation 6 i.e.,

$$\max_{\mathbf{\Theta}} \int P^{test}(x) \log\Big(\hat{h}\,(x;\mathbf{\Theta})\Big)dx \approx \max_{\mathbf{\Theta}} \frac{1}{N}\sum_{i=1}^{N} \log\Big(\mathbf{\Theta}^T \mathbf{K}(x_i^{test})\Big) \tag{7}$$

where the definition of $\hat{h}$ from Equation 4 is used. The constraint to the above equation is based on total probability being 1 i.e., $\int \hat{P}^{test}(x)dx = 1$. Using Equation 4, this can be written as

$$1 = \int P^{ref}(x)\hat{h}(x;\mathbf{\Theta})dx \approx \frac{1}{N}\sum_{i=1}^{N} \mathbf{\Theta}^T \mathbf{K}(x_i^{ref}) \tag{8}$$

It can be shown that Equation 7 together with the constraint Equation 8 is a convex optimization problem, which can be solved using a gradient descent approach. Finally, using Equations 2-5, the statistical distance between the reference and test window is expressed as

$$\begin{aligned} d(W^{test}, W^{ref}) &= \int \Big(P^{test}(x) - P^{ref}(x)\Big) \log\Big(\hat{h}(x;\mathbf{\Theta})\Big)dx \\ &= \frac{1}{N}\sum_{i=1}^{N} \log\Big(\mathbf{\Theta}^T \mathbf{K}(x_i^{test} - x_i^{ref})\Big) \end{aligned} \tag{9}$$

Summary: Following are steps to calculate statistical distance

- Generate two windows W^{test} and W^{ref} with the current and previous workload segment's CPU cycles;
- Find $\mathbf{\Theta}$, by solving the convex problem Equation 7 using constraint Equation 8;
- Calculate the statistical distance using Equation 9.

A. Quality of Change Point Detection

Quality of application change point detection technique is measured in terms of the number of false positives and false negatives. These are defined as:

False Positive: An error state where the change detection algorithm identifies a change point in workload, when in reality there is no such change. A change point resets the learning algorithm (discussed in the next section). Thus, a false positive results in re-learning control levers for an already learned workload resulting in learning-related energy overhead.

False Negative: An error state where the change detection algorithm fails to identify a true change point present in a workload. Thus, a false negative results in a new workload to be processed using control levers learned from an old workload, resulting in both performance and energy overheads.

We combine these error states into a single evaluation function – **Detection Quality**[1], defined as

$$\text{Detection Quality} = 1 - \frac{\#(\text{False +ve}) + \#(\text{False -ve})}{\#(Number\ of\ Changes)} \tag{10}$$

[1]The detection quality is calculated based on the evaluator function of a confusion matrix or the contingency table for a predictive analysis. Refer https://en.wikipedia.org/wiki/Confusion_matrix.

IV. Learning-Based Run-time Manager

Figure 4 shows a three-layer view of an embedded system. The top most layer is the **application** layer, which is composed of active applications. The middle layer is the **operating system** layer (e.g., iOS, Ubuntu, Android, etc), which coordinates an application's execution on the hardware. Finally, at the bottom there is the **hardware** layer, consisting of multicore processors. All three layers interact with each other to execute an application. These interactions are indicated with arrows in the figure. The proposed Run-Time Manager is indicated by the box **RTM** inside the operating system layer.

The RTM, which uses Q-learning algorithm (a variant of reinforcement learning), repeatedly observes the current state of a system, and selects an action. The selected action changes the system's state, which is used to determine the immediate numeric payoff. Positive payoff is termed as profit and negative payoff as punishment. Initially, the RTM does not know what effects its actions have on the state of the system, nor what immediate payoffs its actions will produce. Rather, it tries out various actions in different states computing payoffs, which are stored in a table (the **Q-table**). This phase of the algorithm is known as the **exploration** phase. To make this framework robust, the RTM needs to further evaluate good decisions (those with rewards) by repeatedly selecting them and observing the state of the system. This phase of the algorithm is known as **exploration-exploitation**. In this phase the RTM uses a fraction of the payoff to update the Q-table. Finally, at the end of this phase, the RTM is said to have fully learnt a workload's thermal behavior. This phase is known as the **exploitation** phase, and the RTM always selects the best action (i.e., the action corresponding to the highest payoff) for a particular system state.

The RTM works at the system time ticks (indicated in the figure). It is to be noted that in this work, we adopt a proactive thermal management approach; therefore, the next system state is predicted and appropriate actions are enforced, before the system reaches the state. In this way, the approach prevents thermal emergencies (proactive), rather than reacting when such emergencies occur (reactive). Workload prediction is inherent to this algorithm[2], i.e. at time t_i the algorithm predicts workload for the next interval to select the best action. Specifically, at time instant t_i, the RTM performs following:

- computes payoff for the time interval $t_{i-1} \rightarrow t_i$;
- updates the Q-table entry corresponding to the state and action at time t_{i-1};
- predicts the system state for interval $t_i \rightarrow t_{i+1}$;
- selects the action for the interval $t_i \rightarrow t_{i+1}$ based on the predicted state.

The mapping of different components of the Q-learning-based run-time approach are discussed next.

[2]In the proposed approach we use both prediction and detection. While prediction is used to predict the system state allowing proactive thermal management for an individual application phase, detection is used to detect application phase change in order to perform application phase-specific thermal optimization. The flow of the approach is as follows: the detection algorithm (change-point detection) is used to detect a phase change; subsequently, the prediction algorithm (EWMA) is used to perform proactive thermal management for the detected phase.

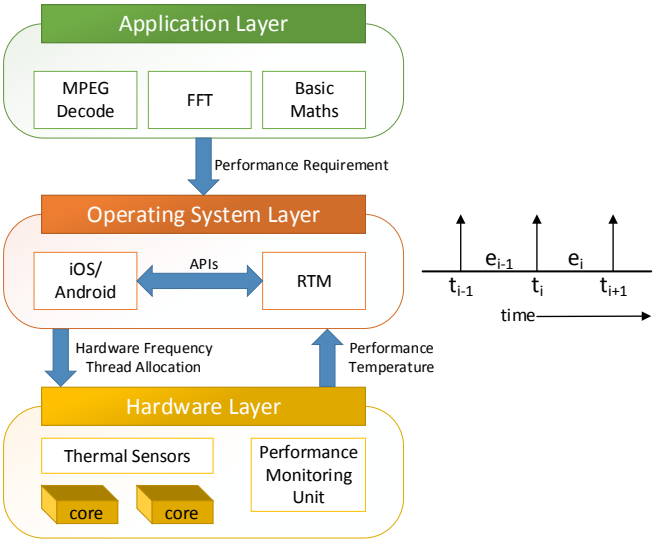

Fig. 4. Representing an embedded systems as three stacked layers – application, operating system and the hardware.

Payoffs: Payoff defines the optimization objective, which in our context is the thermal overhead (average temperature, peak temperature and thermal cycling, combined using Equation 1). Since we are concerned with a constrained optimization problem, the performance constraint needs to be incorporated in the payoff, which is given by the following equation

$$P(t_i) = \begin{cases} w_t \times [K - T_O(t_{i-1} \rightarrow t_i)] & \text{if } F_i \geq F_c \\ w_s \times (F_i - F_c) & \text{otherwise} \end{cases} \quad (11)$$

where $P(t_i)$ is the payoff calculated at time instance t_i, F_i is the application performance during the interval t_{i-1} to t_i, $T_O(t_{i-1} \rightarrow t_i)$ is the thermal overhead of the system in this interval (Equation 1), F_c is the performance constraint, K is a constant, and w_t and w_s are respectively the weights for the temperature and performance. The performance of an application is the inverse of its timing requirement. The equation is interpreted as follows: if the performance obtained in the interval of interest is greater than the performance constraint, the thermal overhead is used to compute the payoff. On the other hand, if there is performance violation, the negative of the performance slack is used as the payoff to prevent the system from reaching this state again in the future.

System State: The state of an embedded system is represented using CPU cycles obtained by reading the PMU. However for some systems, direct accesses to performance registers are disabled in the user mode of operation. For such systems, CPU utilization can be used as an alternative [12]. Thus, the system state s_i at time t_i is given by

$$s_i = Statistics(t_{i-1} \rightarrow t_i) \quad (12)$$

where $Statistics(t_{i-1} \rightarrow t_i)$ is the performance-related statistics (CPU Cycles or utilization) in the interval $t_{i-1} \rightarrow t_i$. CPU Cycles or utilization is a real number in the interval 0 to MAX. To limit the state space, each state s_i is discretized to one of the N_s levels and is indicated as $\hat{s}_i$. These discrete states form rows of the Q-table.

System Action: The action space comprises of the thermal control levers – processor voltage-frequency (hardware lever) and thread affinity (software lever), similar to [3]. Let the affinity be represented as a matrix:

$$M_a(k) = \begin{pmatrix} c_1^k & c_2^k & \cdots c_{N_t}^k \end{pmatrix} \quad (13)$$

where N_t is the number of threads, c_j^k is the core where thread j is allocated in the k^{th} configuration and $c_j^k \in \{c_1, c_2, \cdots, c_{N_c}\}$ with N_c being the number of cores. Most embedded and high performance systems allow chip-wide DVFS, i.e. all processing cores are in the same voltage domain, allowing a single setting for all cores. Therefore, the k^{th} action can be represented as

$$a_k = \langle M_a(k) \;||\; (V_k, f_k) \rangle \quad (14)$$

i.e. an action is composed of the thread affinity matrix and the voltage-frequency value for all cores. Here, $(V_k, f_k) \in \{(V_1, f_1), (V_2, f_2), \cdots, (V_{N_f}, f_{N_f})\}$, the N_f voltage-frequency pairs supported on the hardware. Usually, OS allows scaling the frequency using the `cpufreq` API. The voltage is scaled proportionately. Therefore, Equation 14 can be simplified to

$$a_k = \langle M_A(k) \;||\; f_k \rangle \quad (15)$$

These actions form columns of the Q-table. The total number of actions of the Q-learning is given by

$$N_a = N_f \times N_c^{N_t} \quad (16)$$

Clearly, the number of actions grows exponentially with an increase in the number of threads and cores. Later in Section IV-A, we discuss algorithmic modifications to limit the number of actions of the Q-learning algorithm.

Q-table Update: The Q-table entry corresponding to a state-action pair at time t_{i-1} is updated at time t_i, using the payoff as given below.

$$Q(\hat{s}_{i-1}, \hat{a}_{i-1}) = Q(\hat{s}_{i-1}, \hat{a}_{i-1}) + \alpha \times P(t_i) \quad (17)$$

where $\hat{a}_{i-1} \in \{a_1, \cdots, a_{N_a}\}$ is action during time $t_{i-1} \rightarrow t_i$, α $(0 \leq \alpha \leq 1)$ is the learning rate and represents fraction of the payoff used as learning experience for updating the Q-table entry. This is computed as

$$\alpha = \begin{cases} 1 & \text{for } 0 \leq N_v < N_{explore} \\ 2^{(N_{explore} - N_v)} & \text{for } N_{explore} \leq N_v < N_{exploit} \\ 0 & \text{for } N_v \geq N_{exploit} \end{cases} \quad (18)$$

where N_v is the number of visits, and $N_{explore}$, $N_{exploit}$ are the constants indicating the limits of the Q-learning stages, i.e. exploration and exploitation. These parameters are selected considering the trade-off between Q-table convergence rate and exploration time. For a balance of these metrics, we have used $N_{explore} = 3$ and $N_{exploit} = 8$ (similar to of [3]).

Action Selection: As discussed before, the RTM selects an action at time t_i for controlling the thermal overhead in the time interval $t_i \rightarrow t_{i+1}$ (proactive approach). So, the RTM first needs to predict state of the system for interval $t_i \rightarrow t_{i+1}$; subsequently, the RTM selects an action that has previously resulted in the least thermal overhead for that state. To effectively predict the system state, we use the

TABLE II
MEMORY AND ENERGY OVERHEAD OF THE Q-TABLE WITH FOUR HARDWARE FREQUENCIES ON DUAL-CORE ARM A9.

# Threads	Non-hierarchical		Hierarchical	
	Q-table related memory (KB)	Q-table related retention power (mW)	Q-table related memory (KB)	Q-table related retention power (mW)
2	0.625	0.15	0.15	0.04
4	2.500	0.62	0.15	0.04
6	5.625	1.40	0.15	0.04
8	10.00	2.50	0.15	0.04

exponential weighted moving average (**EWMA**) technique. In this technique, the predicted system state p_{i+1} during the time interval $t_i \rightarrow t_{i+1}$ is given by

$$p_{i+1} = \gamma \times s_i + (1 - \gamma) \times p_i \tag{19}$$

where γ is the smoothing factor. The equation is interpreted as follows. The predicted state in time interval $t_i \rightarrow t_{i+1}$ is determined from the predicted state during interval $t_{i-1} \rightarrow t_i$ (i.e. p_i) and also, the actual state during this interval (i.e. s_i). The action selected for this interval is given by

$$a_{i+1} = \texttt{argmax}\ \text{Q-table}(\hat{p}_{i+1}, :) \tag{20}$$

where Q-table$(\hat{p}_{i+1}, :)$ is the Q-table row corresponding to the predicted state p_{i+1} (discretized to $\hat{p}_{i+1}$) and `argmax` returns the index of the highest argument.

A. Reducing the Action Space

As shown in Equation 16, the number of actions increases exponentially with the number of threads and cores. This also increases (1) the size of the Q-table required to store learning values; (2) the power required to retain the Q-table values in RAM; and (3) the time required for the algorithm to learn the thermal behavior, before applying appropriate thermal control levers. Memory and power overhead are reported in Table II (columns 2-3) for different thread combinations on the experimental platform with a dual-core ARM A9 CPU.

The power overhead is computed as follows. The total number of entries in the Q-table = $N_a \cdot N_s$. Each entry is represented using 8 bits. The power needed to retain a bit is obtained using [13]. The non-hierarchical (flat) approach of integrating thread affinity and frequency selection as part of the same Q-table results in exponential growth of the table size and a corresponding increase in the power required to retain Q-table values as the number of threads increases. It is interesting to note that with 8 threads, the power consumption is as high as 2.5 mW. The power consumption also increases linearly as the number of frequency levels increases.

To prevent state space explosion and providing a reasonable Q-table size, we adopt a two-level hierarchical approach: heuristic-based thread affinity selection, and learning-based frequency scaling. Thread affinity is selected at a higher interval, called the **Thread affinity Selection Interval** (`TSI`). Frequency is scaled at a shorter interval, called the **Frequency Selection Interval** (`FSI`). The choice of `TSI` and `FSI` are justified in Section V.

ALGORITHM 1: Pseudo-code of the hierarchical RTM

Input: Temperature T obtained during decoding the previous frame, frame count cnt
Output: Thread affinity array M_a and hardware frequency f

1 Initialize $M_a[i] = i(\texttt{modulo})N_c$; $TO = CalcThermalOverhead$;
2 Initialize $rerun = 1$; $tid = cid = 0$ and $M_a[tid] = cid$;
3 $TA01.\texttt{push}(T)$ and $TA02.\texttt{push}(T)$;
4 /* Greedy heuristic-based affinity selection */;
5 **if** $cnt\ \%\ TSI == 0$ *and* $rerun == 1$ **then**
6 $\quad TempArr[cid] = CalcThermalOverhead(TA01)$;
7 $\quad$ Calculate performance F_i;
8 $\quad$ **if** $F_i < F_c$ **then** $TempArr[cid] = \infty$;
9 $\quad cid = (cid + 1)\%N_c$;
10 $\quad M_a[tid] = cid$;
11 $\quad$ **if** $cid == 0$ **then**
12 $\quad\quad C = \texttt{argmin}(TempArr)$;
13 $\quad\quad M_a[tid] = C$;
14 $\quad\quad tid = (tid + 1)\%N_t$;
15 $\quad$ **end**
16 $\quad$ /* Termination of the greedy algorithm */;
17 $\quad$ **if** $tid = cid = 0$ **then**
18 $\quad\quad$ **if** $CalcThermalOverhead(TA01) < TO$ **then**
19 $\quad\quad\quad TO = CalcThermalOverhead(TA01)$;
20 $\quad\quad$ **end**
21 $\quad\quad$ **else**
22 $\quad\quad\quad rerun = 0$;
23 $\quad\quad$ **end**
24 $\quad$ **end**
25 $\quad TA01.\texttt{clear}()$;
26 **end**
27 /* Q-learning based frequency selection */;
28 **if** $cnt\ \%\ FSI == 0$ **then**
29 $\quad$ Calculate Payoff (Equation 11);
30 $\quad$ Update Q-table entry (Equation 17);
31 $\quad$ Predict Next State (Equation 19);
32 $\quad$ Select Action (Equation 20);
33 $\quad$ Map action to hardware frequency;
34 $\quad TA02.\texttt{clear}()$;
35 **end**

B. Hierarchical RTM Algorithm

Algorithm 1 provides the pseudo-code of the hierarchical RTM, where affinity selection is performed using a greedy heuristic (lines 1-26) and frequency selection is performed using the Q-learning algorithm (lines 28-35), which is the simplified version of the learning algorithm used in our earlier work [3].

An array M_a is used to store the affinity matrix, i.e. $M_a[i]$ stores the core id where thread i is mapped. At the start of the algorithm, threads are distributed equally on the cores, i.e. $M_a[i] = i(\texttt{modulo})N_c$. This balances the number of threads on the cores. The thermal overhead using this balanced thread distribution is computed and stored in a local variable TO. Subsequently, the greedy algorithm is triggered. Two running variables tid and cid are used to hold the values of the current thread and core, respectively. These are initialized in line 2 of the algorithm. The affinity matrix is changed by allocating thread 0 on core 0. It is to be noted that another local variable $rerun$ is initialized to 1 and is used to determine the termination of the greedy algorithm.

The algorithm takes the previous frame's temperature as an input. This temperature is pushed in two arrays $TA01$ and $TA02$ for use in affinity selection and frequency selection sub-processes, respectively. At every TSI, the algorithm calculates the thermal overhead (using Equation 1) from the thermal

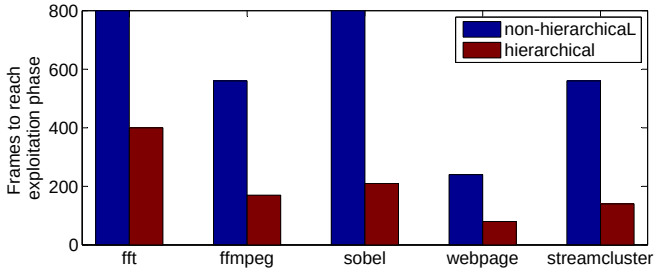

Fig. 5. Number of frames needed to reach steady-state (the exploitation phase) for non-hierarchical (flat) vs hierarchical approach.

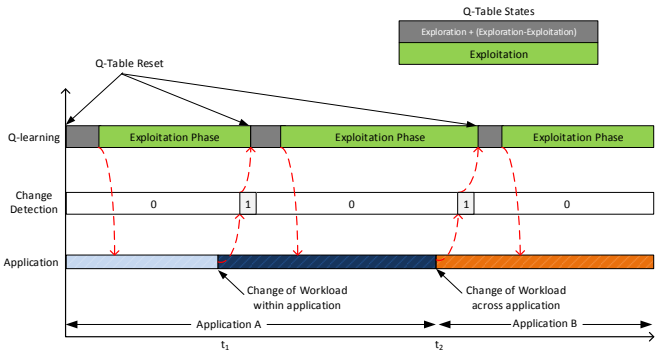

Fig. 6. Timing diagram, showing handshaking between Q-learning and change point detection.

sensor samples (line 6). This is stored in the array $TempArr$ corresponding to the currently selected core. The application performance is determined (line 7). If this performance is lower than the performance constraint F_c (i.e., the performance is violated), the array entry $TempArr[cid]$ is replaced by infinity or a very large number (line 8), such that this thread-core selection is avoided in the subsequent step. The core id is incremented; a core id of zero indicates that all core combinations for the current thread has been explored. The core id with the least thermal overhead is selected (line 12) and the assignment is permanently changed (line 13). The thread id is incremented to repeat the process for the next thread.

The greedy algorithm is terminated using the local variable TO as shown in lines 17 - 24. A thread id and the core id of zero indicates that the algorithm has reached one complete iteration of thread-core selections. After this iteration, the thermal overhead is compared with the initial thermal overhead TO. If the thermal overhead is reduced, then the local variable TO is updated with the new value of the thermal overhead, and an another iteration is initiated. Otherwise, the $rerun$ variable is reset causing the algorithm to terminate.

Frequency selection is performed using the Q-learning algorithm (discussed in details in Section IV).

C. Scalability of the Hierarchical Approach

Table II (columns 4-5) reports the memory and power required for the Q-table entries using the hierarchical approach on the same experimental platform and with the same setup. It is to be noted that, for the hierarchical approach, the Q-table stores only the frequencies as the actions. Therefore, the size and the retention power are both independent of the number of threads. For 8 threads, the proposed approach results in a 64x reduction of Q-table size and a similar reduction in power consumption. To evaluate the scalability of the hierarchical approach, Figure 5 plots the number of frames needed by the learning algorithm to reach the exploitation state for five applications executed on the PandaBoard with a dual-core ARM A9 CPU (Table IV). The number of frames to reach the exploitation stage is an indication of the convergence time of the algorithm. As can be seen from this figure, the hierarchical approach results in a lower convergence time than the non-hierarchical one. This is because thread-affinity selection is performed using a greedy search as opposed to being part of the Q-table based exploration in the non-hierarchical technique. On average, the hierarchical approach reduces convergence time by 3.2x, signifying the scalability.

D. Implementation Details

Figure 6 shows a timing diagram illustrating the execution of two applications – application A followed by the execution of application B. Application A has a workload change during its execution (at time t_1) and application B has a relatively constant workload. The switch from application A to B takes place at time t_2. The Q-learning algorithm and the change point detection algorithm are implemented as separate threads and are allowed to be executed on any core alongside the application threads (controlled using thread affinity). As seen in the figure, the workload change of A at time t_1 is detected using the change detection thread (indicated by a value of *1* in the timing diagram); the Q-learning thread is signaled to reset the Q-table in order to restart the learning process. A similar sequence of events follow when there is a switch of application (at time t_2).

Communication between these threads are implemented as message queues. There are two queues implemented – `frameQ` and `changeQ`. The `frameQ` is a signal from the application thread to the Q-learning thread indicating the completion of a frame. This is used along with a hardware timer to compute the frame rate, which is then used to determine the performance impact of hardware and software control levers (refer to Section IV). The `ChangeQ` is a signal from the change detection thread to the Q-learning thread, forcing the Q-table to reset. The inter-process communication delays using the message queues are illustrated in the figure with red dashed lines.

V. Results

Experiments are conducted with CPU-intensive and embedded applications to demonstrate the proposed approach. The CPU-intensive applications are taken from PARSEC and the SPLASH2 [14] benchmark suites, and embedded ones from MiBench [15]. Additionally, a few mobile applications are also considered. Applications used for validation of our approach are listed in Table III. Each application is transformed to a periodic structure, where the application is executed for several iterations; each iteration is accompanied by a deadline,

TABLE III
APPLICATIONS USED FOR DIFFERENT EXPERIMENTS.

Category	Suite	Benchmarks
High Performance	PARSEC	blackscholes,bodytrack,fluidanimate,swaptions,streamcluster
	SPLASH 2	tachyon, x264
Mobile	openCV	sobel filter
	MiBench	fft,mpegenc,basicmath,jpeg,tiffmedian,gsm
	Video Player	ffmpeg
	Browsing	webpage

TABLE IV
PLATFORMS USED FOR DIFFERENT EXPERIMENTS.

Platform	SoC	Mobile Use	Cores
PandaBoard	TI OMAP	GALAXY nexus	Dual-core ARM A9
Jetson TK1	NVIDIA Tegra	HTC Touch	Quad-core ARM A15
Odroid XU3	Samsung Exynos	Samsung S5	Quad-core ARM big.LITTLE (A7-A15)

which serves as the performance requirement. At each iteration multiple threads are spawned, with each thread performing some task on the input data. These iterations are referred to as frames throughout the remainder of this work. It is important to note that video applications (ffmpeg, openCV.sobel etc.) automatically align to this general structure with a frame representing a video picture or a group of pictures (GoPs). These applications are executed on three ARM-based embedded platforms, each running Linux. Characteristics of the SoCs for these platforms are reported in Table IV. Unless otherwise stated, all experiments are conducted on the Jetson platform; however, thermal results on the different platforms are provided in Table V to demonstrate the consistency of our result across different platforms. The scope of this work is limited to homogeneous systems. GPUs present in some of the experimental boards are forced into sleep mode by the operating system to minimize their thermal impact on CPU.

Temperature results are directly recorded from on-board thermal sensors, which intrinsically incorporates the RC thermal behavior of the system. Additionally, through the machine learning we learn the relationship between input workload and output thermal response. In all experiments, we have disabled default thermal management policies of the operating system to fairly estimate the thermal improvements.

A. Thermal Improvement Across Different Applications

Figure 7 plots three thermal parameters for five applications – three mobile (fft, x264 and openCV) and two CPU intensive ones (streamcluster and swaptions). Results obtained using the proposed technique are compared with two state-of-the-art approaches – Linux's default **ondemand** governor [16] and the learning-based technique [17] of Ge et al. A common trend to follow from this figure is that average and peak temperatures are higher for CPU intensive applications compared to mobile applications. This is expected because CPU intensive applications demand high processing power, resulting in high CPU utilization and elevated temperatures.

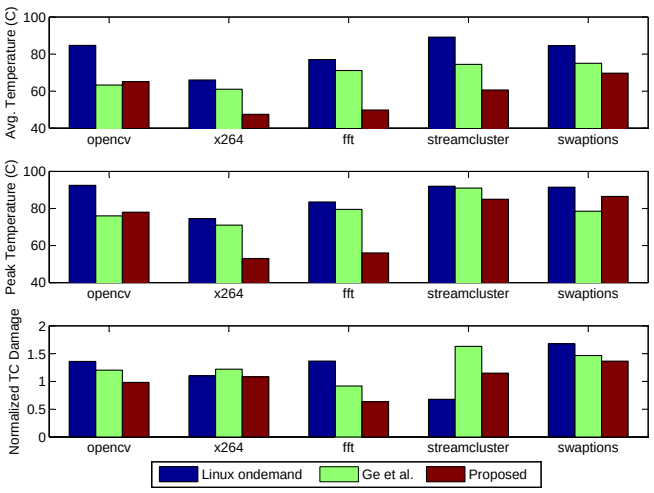

Fig. 7. Thermal improvement of the proposed approach on Jetson TK1.

As seen from the figure, the technique of Ge et al. improves average and the peak temperatures significantly compared to the ondemand governor. On average, this technique minimizes average temperature by 10°C and peak temperature by 6°C. These improvements are due to the consideration of peak temperature in the optimization objective, which also improves the average temperature. It is important to note that, due to the consideration of peak temperature, this technique achieves the lowest peak temperature for some applications such as the swaptions. However, this technique does not guarantee reducing thermal cycling. This can be seen for the streamcluster application, where the technique of Ge et al. has increased thermal cycling related damage by more than 140% compared to the ondemand governor.

In comparison to these state-of-the-art approaches, the proposed approach balances all thermal parameters. For some applications such as the fft, the proposed approach outperforms both these state-of-the-art approaches in terms of all three thermal parameters. Specifically, for this application, the proposed approach reduces average temperature by 27.2°C, peak temperature by 27.5°C and thermal cycling by 54% with respect to the ondemand governor. In comparison to the technique of Ge et al, improvements in these thermal parameters are 21.3°C, 23.5°C and 31%, respectively. For applications such as the streamcluster, average temperature using the the ondemand governor is 90°C. This high temperature degrades platform reliability, increases power consumption and negatively impacts performance by throttling the CPU. These limitations are addressed in the proposed technique and also in the technique of Ge et al. While the technique of Ge et al. have reduced the average temperature to 74.5°C (reduction by 14.5°C), thermal cycling is increased by 140%. The proposed two stage hierarchical approach controls both these parameters efficiently; reducing average temperature further to 60.6°C (a further reduction of 13.8°C) and thermal cycling by 30% with respect to the technique of Ge et al.

Summary: On average for all applications considered (including those not shown in Figure 7), the proposed approach reduces average temperature by 4–25°C, peak temperature

TABLE V
THERMAL IMPROVEMENT OF THE PROPOSED APPROACH OVER LINUX'S DEFAULT ONDEMAD GOVERNOR FOR DIFFERENT PLATFORMS. EACH ENTRY REPORTS THE IMPROVEMENT IN AVERAGE TEMPERATURE (THERMAL CYCLING).

Application	Data Set	PandaBoard	Jetson TK1	Odroid XU3
tachyon	set 1	18.6°C (-29%)	18.1°C (4.2%)	20.1°C (7.8%)
	set 2	6.7°C (47.2%)	12.1°C (55.9%)	12.5°C (61.0%)
	set 3	9.2°C (80.0%)	11.9°C (133.2%)	12.2°C (130.0%)
ffmpeg	clip 1	1.8°C (67.2%)	14.1°C (81.2%)	13.7°C (101.1%)
	clip 2	1.4°C (76.6%)	3.1°C (96.4%)	6.3°C (112.2%)
	clip 3	0.3°C (56.7%)	9.6°C (88.8%)	13.2°C (100.8%)
mpeg enc	seq 1	1.1°C (17.3%)	16.2°C (23.3%)	20.8°C (51.2%)
	seq 2	2.1°C (18.7%)	17.5°C (21.1%)	15.7°C (39.4%)
	seq 3	1.4°C (9.8%)	15.1°C (11.1%)	18.9°C (22.6%)

by 6–24°C, and thermal cycling related damage by 7–35% compared to the existing technique of Ge et al.

B. Thermal Improvement Across Different Platforms

To demonstrate consistency of the proposed approach across different embedded platforms, we conducted an experiment with different applications, comparing the average temperature and the thermal cycling obtained using our approach with the ondemand governor. For the tachyon application on data set 1, the proposed approach reduces average temperature by 18.6°C compared to the ondemand governor on the PandaBoard with dual core A9 CPU. However, the approach increases thermal cycling by 29%. On quad core A15 CPU-based Jetson TK1, the proposed approach reduces average temperature by 18.1°C and reduces thermal cycling by 4.2%. It is interesting to note the improvement in thermal cycling in moving from dual core to quad core. This is because, with quad-core system, the affinity lever has more state-space to explore than a dual core system, achieving better results. A similar trend is observed for Odroid XU3 with 8 cores. It is to be noted that the table reports average temperature and thermal cycling; the improvement in peak temperature is on a similar scale as the average temperature.

Summary: For all applications and data sets considered, the proposed approach outperforms existing techniques in all three thermal aspects. Furthermore, the improvement using our approach increases with more number of cores, demonstrating its suitability for future many-core systems.

C. Performance-Thermal Trade-off

Figure 8 plots the time taken for decoding each frame of a reference 1080p video played using the ffmpeg application. The application is configured to drop frames if the decoding time exceeds a fixed threshold. As seen in this figure, a frame is dropped when the decoding time exceeds 50ms for a 24 fps video. This threshold is set by the ffmpeg applications and can be overridden in the user mode. There are 17 frames being dropped during decoding of 260 frames. This is because the

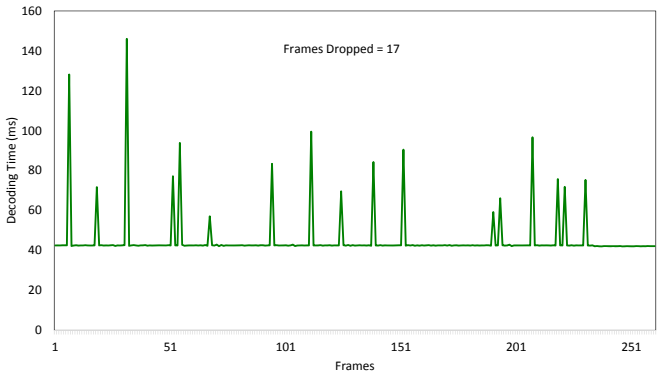

Fig. 8. Performance slack for a 24 fps 720p video decoded using the ffmpeg application. Two scenarios are reported: (a) performance slack without change point detection and (b) performance slack with change point detection.

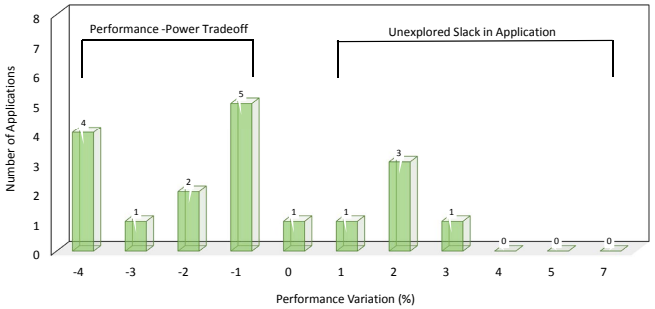

Fig. 9. Performance summary across 20 different applications.

proposed run-time manager reduces the frequency of operation for these frames to mitigate the increase in thermal aspects. For all decoded frames, the average decoding time is 42.3 ms instead of 41.67 ms for a 24 fps video. This increase in decoding time results in performance loss (1.5%), however gaining significantly on thermal parameters (13.2°C reduction in average temperature and over 80% improvement in thermal cycling, ref. Table V, column 7).

To summarize the results for other applications, we conducted experiments with twenty different applications from the benchmark suites previously discussed. Figure 9 shows a performance summary for these applications. The x-axis of this figure reports the percentage performance variation using the proposed approach (with respect to the specified deadline). The length of each bar represents the number of applications with the corresponding violations. In representing the number of applications, we used a ceiling function. As an example, the ffmpeg application has a steady-state performance violation of 1.52% and is represented along with other applications as part of the bar corresponding to a violation of -2%. It is important to note that 70% of applications (14 out of 20) have negative performance variations implying that, for these applications, the proposed approach achieves thermal improvements by trading less than 5% in performance. There are 6 applications which have positive performance variations, i.e. for these application the proposed approach is not able to exploit remaining application slack for thermal improvement opportunities. The highest performance slack that remains to

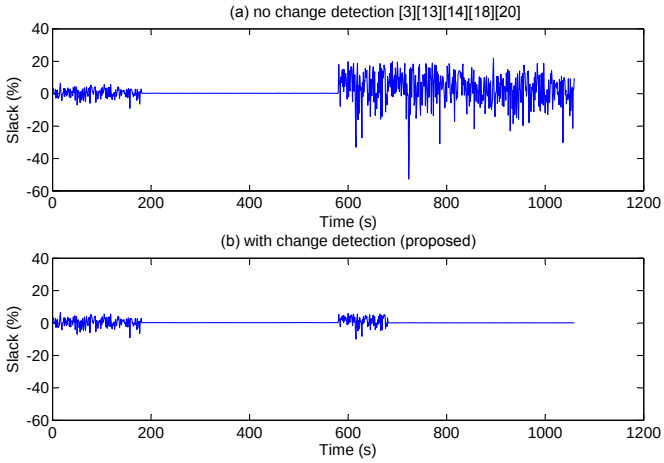

Fig. 10. Performance slack for a 24 fps 720p video decoded using the ffmpeg application. Two scenarios are reported: (a) performance slack without change point detection and (b) performance slack with change point detection.

be exploited is 3% (in the figure, the number of application with performance variation of 4% or above is zero).

D. Impact of Change Point Detection on Performance

As highlighted before, in the context of change point detection, false negative has high performance penalty. To highlight this, an experiment is conducted with the ffmpeg decoder playing a 720p video encoded at 24 fps. Figure 10 shows performance slack for two scenarios: (a) without change point detection (such as [17], [3]) and (b) with the proposed change point detection approach. In both scenarios, the Q-learning-based run-time approach learns thermal control levers for the video workload during the initial phase (1–80 ms). After this, the learning algorithm reaches the exploitation phase where good actions are selected to satisfy performance (24 fps) and minimize thermal overhead (not shown explicitly in the figure). At around 550 ms, there is a change in workload due to change in video scenes. From this point forward, the two scenarios differ significantly. Without change point detection, the corresponding Q-learning-based run-time approach applies thermal control levers leaned from the initial workload on this new workload. This results in high performance violation for the new workload, with violations as high as 50% reported for some frames. It is to be noted that the ffmpeg application is configured to drop frames if there is performance violation. As a result, there are significant jitters in the video.

With change point detection, the proposed Q-learning-based run-time approach restarts learning all thermal control levers at 550 ms; 150 ms later (i.e. at 700 ms), the proposed run-time approach again reaches the exploitation phase. At this phase, good actions are selected for this new workload and there is no performance violation. This demonstrates the significance of change point detection for thermal optimization, which is not addressed by any existing techniques.

E. Parameters for Change Point Detection

As discussed in Section III, there are two parameters for the change detection technique. These are the size of the sliding window, N and overlapping depth b. An increase in N increases detection quality, however at the expense of increase in the latency, i.e. delay in detecting a change. To demonstrate this, Figure 11 plots the latency vs accuracy trade-off for the density ratio-based change detection technique applied to detect changes in the ffmpeg application playing a 1080p video. As can be seen, with an increase in N, the detection latency increases exponentially. For all experiments in this work, we choose $N = 14$. This gives a detection latency of less than 40 ms (less than a frame for 24 fps video). Similar experiments are conducted for the parameter b, where the detection quality increases with an increase in b, saturating at $b = 5$. We used this value for all experiments.

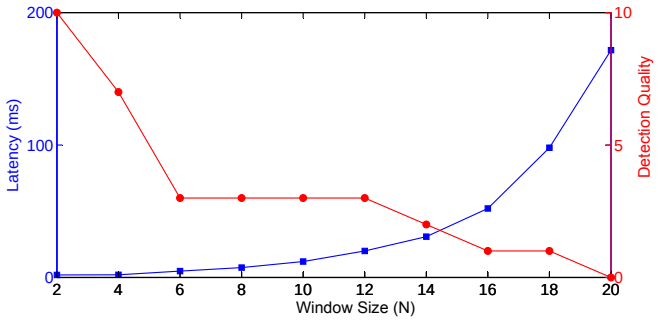

Fig. 11. Latency vs detection quality trade-off.

TABLE VI
DETECTION QUALITY OF DIFFERENT CHANGE POINT DETECTION APPROACHES (EQUATION 10).

Application	Data Set	CPU Cycles Difference	Manhattan Distance	Kullback Leibler Divergence	Improvement	Ref [7]
tachyon	set 1	0.1	0.44	0.80	1.8x	0.98
	set 2	-0.04	0.4	0.84	2.1x	0.85
	set 3	0.02	0.37	0.72	1.9x	0.73
mpeg dec	clip 1	0.09	0.13	0.81	6.2x	0.88
	clip 2	0.15	0.25	0.74	2.9x	0.66
	clip 3	0.02	0.22	0.76	3.4x	0.70
mpeg enc	seq 1	-0.01	0.12	0.77	6.4x	1.00
	seq 2	-0.01	0.33	0.81	2.4x	0.71
	seq 3	-0.04	0.25	0.86	3.4x	0.72

F. Detection Quality of Change Point Detection Approaches

Table VI reports the detection quality of the proposed Kullback-Leibler based change point detection algorithm compared with Manhattan Distance-based and CPU cycles difference-based detection techniques, which are used in existing works [3] [18] (refer to Appendix B). The detection quality is computed using Equation 10. From the definitions of false positive and false negative (Section III) it can be reasoned that the number of false negatives is always less than or equal to the number of change points. However, there is no such constraints on the number of false positives, meaning that the total number of false positives and false negatives can be greater than the total number of change points, resulting in negative value of the detection quality. This

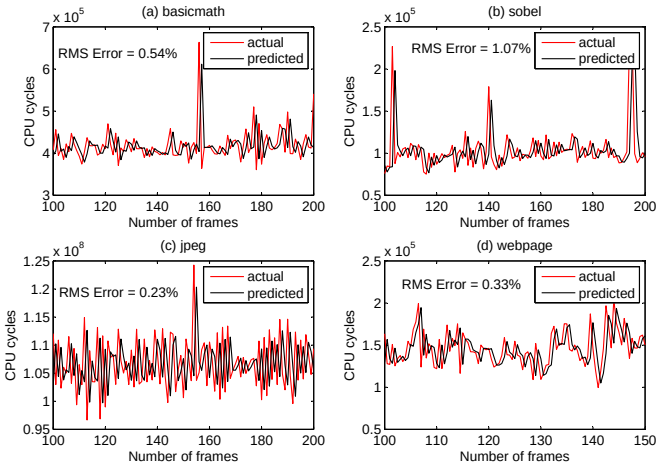

Fig. 12. Workload prediction using EWMA.

is seen for the CPU cycles difference-based detection approach on the mpeg encoding application for all three data sets. The detection quality of Manhattan distance-based approach is higher than the difference-based approach. The Kullback Leibler divergence used in the proposed approach results in the highest detection quality with an improvement of 1.8–6.4x (average 3.4x) compared to Manhattan distance-based approach.

Finally, column 7 of the table reports the detection quality using the multi-variable offline characterization technique of [7]. For this, the first set of each application (set 1 /clip 1 /seq 1) is part of the training data sets used in the principle component analysis process. As a result, the detection accuracy for these sets is high. However, when the offline characterization approach is applied to unknown sets at run-time (set 2/set 3), the detection quality is lower than that of the proposed single variable-based approach using Kullback Leibler divergence for the mpeg enc and mpeg dec. For the tachyon application, results are comparable. Thus, multi-variable offline characterization approach results in higher detection quality for some application and data sets. However, higher quality is not always guaranteed. Compared to this, the proposed single-variable approach is a run-time only approach: no training is needed and yet delivers over 70% detection quality for all applications and data sets.

G. Evaluation of EWMA as the Prediction Scheme

One of the important aspects of the proposed Q-learning-based run-time approach is its proactive nature, i.e. preventing a system from reaching thermal emergencies. Accuracy of this approach is dependent on accurate prediction of future states using EWMA. To demonstrate this, Figure 12 plots the difference between the actual and the predicted workload for four applications: basicmath and jpeg from the MiBench benchmark suite, sobel filtering (an OpenCV application commonly used in mobile phones for face recognition) and webpage (a webpage rendering application commonly executed on an embedded system). In this figure, the actual workload is plotted in red and the predicted one in black for these four

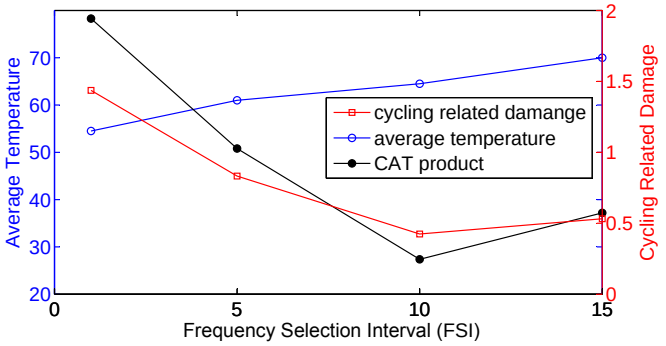

Fig. 13. Frequency selection interval for the freqmine application.

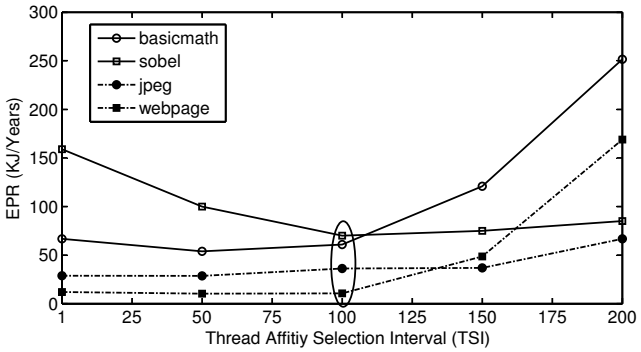

Fig. 14. Selection of thread affinity interval.

applications. The root mean square (RMS) error in workload prediction is also reported (as percentage) for each of these applications. As can be seen, the EWMA-based prediction is able to predict CPU cycles accurately, justifying its choice for the proposed proactive thermal management approach.

H. Selection of Control Intervals

The frequency selection interval (FSI) plays an important role in determining the average temperature and the thermal cycling related damage. To demonstrate this, Figure 13 plots the average temperature (in blue) and the thermal cycling (in red) for the freqmine, a data mining application. As seen from this figure, choosing a low FSI (say 1) results in higher thermal cycling but allowing finer control over temperature. On the other hand, choosing a high FSI (say 15) results in lower thermal cycling but at the expense of higher average temperature. The figure also plots the product of thermal cycling and average temperature (in black) as identified in the figure with the legend *CAT product*, which represents the average thermal stress (Table I, row 6). It can be seen, the product of the two thermal parameters first decreases and then increases again (due to the conflicting nature of the two thermal parameters as discussed). An FSI of 10 results in the minimum value of the thermal stress.

To illustrate the impact of changing the Thread affinity Selection Interval (TSI) on energy and reliability, a joint metric, energy per unit reliability (EPR), is introduced. EPR represents the energy overhead (in KJ) incorporated in the system per unit increase in reliability measured as

TABLE VII
PARAMETER SELECTION FOR DIFFERENT APPLICATION.

Application	Data Set	Window Size N	Overlapping Depth b	Smoothing Factor γ	TSI	FSI
tachyon	set 1	20	5	0.6	110	11
	set 2	15	6	0.6	120	11
	set 3	15	6	0.7	100	8
ffmpeg	clip 1	14	5	0.7	120	10
	clip 2	14	5	0.6	100	8
	clip 3	14	5	0.6	100	10
swaptions	data 1	10	5	0.7	110	10
	data 2	12	6	0.6	110	9
	data 3	16	5	0.6	110	8
basicmaths	data 1	14	5	0.7	100	10
	data 2	10	5	0.7	100	8
	data 3	15	5	0.6	90	10
streamcluster	data 1	15	5	0.7	90	10
	data 2	14	6	0.8	90	10
	data 3	14	5	0.8	110	10
Selected		14	5	0.7	100	10

the mean time to failure (MTTF). This is calculated as $EPR = Energy(KJ)/MTTF(Years)$, i.e., by dividing the energy consumed per iteration of the application by the MTTF obtained assuming the application is executed infinitely on the system. As discussed in Section II, the thread selection interval influences primarily thermal cycling-related aging. Therefore, thermal cycling-related MTTF is only considered to compute the EPR. Figure 14 plots the EPR obtained using the proposed run-time manager for the same four applications, with TSI varying from 1 to 200. When TSI is small, there is frequent switching of application threads. This increases the timing overhead and degrades performance. To meet the performance requirement, the proposed Q-learning-based run-time approach raises the hardware frequency, and therefore the energy consumption is higher for lower TSI. On the other hand, lower TSI results in finer control on thermal cycling and therefore, the reliability values are also higher. When the TSI is increased, the energy overhead decreases and so does the reliability. When the two objectives are combined (as EPR), the value of EPR first decreases and then starts increasing. This is because initially, the decrease in energy dominates over the decrease in reliability causing a fall in ERP. However, beyond a point, the decrease in reliability starts dominating over that in energy consumption causing the overall EPR to increase. This is the general trend observed for most applications. The value of TSI corresponding to the minimum EPR is to be selected. As seen from the figure, TSI = 100 results in best energy-reliability trade-off for most applications.

I. Parameters Selection for Different Applications

Table VII reports values for different parameters obtained by characterizing different applications and data sets. Only five of these applications are reported in the table. The table also reports the final value selected for our approach.

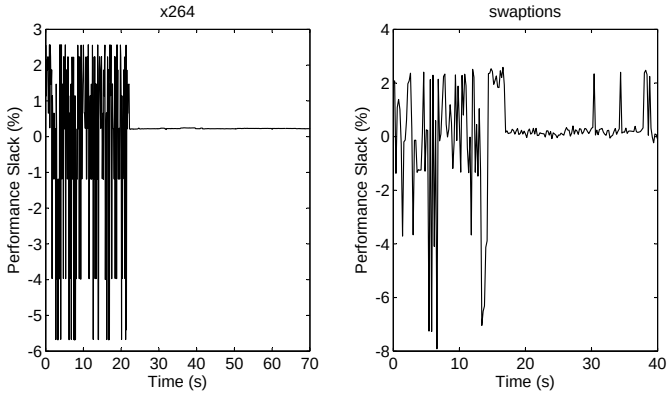

Fig. 15. Performance slack using the proposed algorithm.

J. Performance Obtained at Different Learning Stages

As discussed in Section IV, the Q-learning algorithm of the proposed approach incorporates performance constraint in calculating the payoff (Equation 11), i.e. the Q-learning algorithm controls thread affinity and voltage-frequency scaling such that all thermal parameters are improved while satisfying an application's performance constraint. It is to be noted that, in the exploration phase of the Q-learning algorithm, some selected actions violate the performance requirement. However, as the algorithm reaches the exploitation phase, actions satisfying the performance requirement are only selected. To illustrate this, Figure 15 plots the performance slack for two applications – x264 and the swaptions, executed for a period of time. As can be seen, there are performance violations (negative slacks) during the exploration phase (0-20s) of both these applications with maximum violation of 5.8% and 8%, respectively. For x264 (video playback), a 5.8% violation translates to a decoding time of 44 ms compared to the required 41.6 ms for a 24 fps video. This performance degradation (although not desired) doesn't degrade user experience significantly. In the exploitation phase, however, there is no performance violation.

VI. RELATED WORKS

Thermal optimization has received significant attention recently due to its superlinear dependency on leakage power and lifetime reliability. Some of these works make decisions at design-time, determining application mapping and the voltage-frequency of processing cores so as to minimize temperature-related wear-outs [6], [19], [20]. Contrary to design-time techniques, run-time techniques scale the voltage and frequency of processing cores dynamically in response to the application executing on the system. A thermal prediction model is developed in [2] based on an offline thermal and power characterization of an application. Using this model, optimal and heuristic scheduling techniques are proposed for dynamic thermal management of single instruction set heterogeneous multiprocessor systems-on-chip (MPSoCs). This technique suffers from the following limitations: the accuracy of the offline characterization of an application is dependent on the workload used for the characterization process; the approach is dependent on multiprocessor thermal equivalent models

that are difficult to obtain from manufacturer datasheets and are overly expensive to solve at run-time; thermal cycling is not considered; and the approach is system-dependent and therefore needs to be re-characterized for all MPSoCs, even with the same architecture.

A dynamic thermal management approach is proposed in [21] based on a lumped thermal control model. Using this, an approach is proposed to optimize the performance of an application with soft thermal constraints. This approach also suffers from similar limitations as discussed before. A distributed agent-based approach is proposed in [22] that uses fast context-aware task migration to minimize peak temperature-related hotspots. This approach does not minimize average temperature and thermal cycling. A reinforcement learning-based thermal management approach is proposed in [17] that uses feedback from hardware thermal sensors to adjust the voltage and frequency of processing cores. This work performs performance-aware thermal management using reinforcement learning, similar to our approach. However, decisions are based on thermal sensor readings which are slow to capture thermal changes and therefore, proactive thermal management is not as effective as our approach, which is based on CPU performance counters (refer to Section V for detailed comparison). Additionally, the technique does not adapt to workload-specific variations.

Another learning-based approach is proposed in [23] to manage temperature of multiprocessor systems, and selects between a set of expert policies depending on workload characteristics. HotSpot is used for temperature modeling based on thermal characteristics of UltraSPARC. However, HotSpot has a known limitation on accuracy and simulation time [6], making this approach difficult to use for real-time applications. A control-theoretic approach is proposed in [24] to optimize the lifetime reliability of a multiprocessor system. Task scheduling decisions are controlled at longer intervals and the voltage/frequency scaling is performed at a shorter interval. Another control approach is proposed in [25] to manage the temperature of applications running on multiprocessor systems. A thermal-safe power budgeting is proposed in [26] [27] for dynamic thermal management of many-core system. A fast even-driven approach is proposed in [28] to estimate the temperature of a multiprocessor system. Based on this a thermal aware scheduling approach is proposed to reduce the temperature of the system at run-time. Apart from these works, there are other studies to reduce the power consumption of a multicore system by scaling the hardware frequency dynamically [29]–[32]. However, as shown in [22], these approaches cannot guarantee to minimize a system's thermal overhead effectively for all application.

A common limitation of all these run-time thermal management approaches is that, these approaches cannot detect application changes autonomously. Therefore, thermal control levers cannot be selected optimally for all workload variations. To provide such capabilities, these techniques require either application or user to explicitly indicate such workload changes, complicating the software development.

VII. Conclusions

We propose a Q-learning-based run-time approach for thermal management of embedded systems. This approach is implemented as a hierarchical run-time manager (RTM) for Linux operating system, with thread affinity selected from the upper hierarchy to control thermal cycling and processor voltage and frequency selected from lower hierarchy to control average and peak temperatures. Thread affinity selection is facilitated by a greedy heuristic, while the voltage-frequency selection is facilitated using reinforcement learning algorithm. The overall RTM uses statistical divergence-based change point detection to detect changes in application workload in order to select the most appropriate combination of thermal control levers to manage all thermal aspects efficiently. Extensive evaluation with multimedia and high performance applications on different ARM-based embedded boards demonstrate the advantage of the proposed approach. Our continuing work is to demonstrate this approach with concurrent applications and GPUs. An open source Linux governor implementation of the run-time manager will be made available online for the benefit of the research community.

Acknowledgment

This work was supported in parts by the EPSRC Grant EP/L000563/1 and the PRiME Programme Grant EP/K034448/1 (www.prime-project.org). Experimental data used in this paper can be found at DOI:http://dx.doi.org/10.5258/SOTON/383667.

References

[1] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," in *Proceedings of the International Symposium on Computer Architecture*, 2004.

[2] S. Sharifi, D. Krishnaswamy, and T. Rosing, "PROMETHEUS: A proactive method for thermal management of heterogeneous mpsocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 7, pp. 1110–1123, 2013.

[3] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems," in *Proceedings of the Design Automation Conference*, 2014.

[4] "Failure mechanisms and models for semiconductor devices," *JEDEC*, vol. JEP122G, 2011.

[5] H. Amrouch, V. M. van Santen, T. Ebi, V. Wenzel, and J. Henkel, "Towards interdependencies of aging mechanisms," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2014, pp. 478–485.

[6] A. Das, A. Kumar, and B. Veeravalli, "Reliability and energy-aware mapping and scheduling of multimedia applications on multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, 2015.

[7] R. Cochran and S. Reda, "Consistent runtime thermal prediction and control through workload phase detection," in *Proceedings of the Design Automation Conference*. ACM, 2010.

[8] M. Ghorbani, Y. Wang, Y. Xue, M. Pedram, and P. Bogdan, "Prediction and control of bursty cloud workloads: A fractal framework," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*. New York, NY, USA: ACM, 2014.

[9] V. N. Vapnik and V. Vapnik, *Statistical Learning Theory*. Wiley New York, 1998, vol. 2.

[10] X. Nguyen, M. Wainwright, and M. Jordan, "Estimating divergence functionals and the likelihood ratio by convex risk minimization," *IEEE Transactions on Information Theory*, vol. 56, no. 11, pp. 5847–5861, Nov 2010.

[11] S. Liu, M. Yamada, N. Collier, and M. Sugiyama, "Change-point detection in time-series data by relative density-ratio estimation," *Neural Networks*, vol. 43, no. 0, pp. 72 – 83, 2013.

[12] M. J. Walker, A. K. Das, G. V. Merrett, and B. Hashimi, "Run-time power estimation for mobile ad embedded asymmetric multi-core cpus," *HiPEAC Workshop on Energy Efficiency with Heterogenous Computing*, 2015.

[13] S. Yang, S. Khursheed, B. Al-Hashimi, D. Flynn, and G. Merrett, "Improved state integrity of flip-flops for voltage scaled retention under pvt variation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 11, 2013.

[14] C. Bienia, S. Kumar, and K. Li, "PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors," in *IEEE Symposium on Workload Characterization*, 2008.

[15] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE Workshop on Workload Characterization*, 2001.

[16] V. Pallipadi and A. Starikovskiy, "The Ondemand Governor," in *Proceedings of the Linux Symposium*, 2006.

[17] Y. Ge and Q. Qiu, "Dynamic thermal management for multimedia applications using machine learning," in *Proceedings of the Design Automation Conference*, 2011.

[18] R. A. Shafik, A. K. Das, L. A. Maeda-Nunez, S. Yang, G. V. Merrett, and B. Al-Hashimi, "Learning transfer-based adaptive energy minimization in embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015.

[19] I. Ukhov, P. Eles, and Z. Peng, "Probabilistic analysis of power and temperature under process variation for electronic system design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 6, pp. 931–944, 2014.

[20] B. H. Meyer, A. S. Hartman, and D. E. Thomas, "Cost-effective lifetime and yield optimization for noc-based mpsocs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 19, no. 2, pp. 12:1–12:33, 2014.

[21] B. Shi, Y. Zhang, and A. Srivastava, "Dynamic thermal management under soft thermal constraints," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 21, no. 11, pp. 2045–2054, 2013.

[22] M. A. Faruque, J. Jahn, and J. Henkel, "Runtime thermal management using software agents for multi- and many-core architectures," *IEEE Design & Test of Computers*, vol. 27, no. 6, pp. 58–68, 2010.

[23] A. K. Coskun, T. S. Rosing, and K. C. Gross, "Temperature management in multiprocessor socs using online learning," in *Proceedings of the Design Automation Conference*, 2008.

[24] P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini, "A linux-governor based dynamic reliability manager for android mobile devices," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2014.

[25] F. Sironi, M. Maggio, R. Cattaneo, G. Del Nero, D. Sciuto, and M. Santambrogio, "ThermOS: System support for dynamic thermal management of chip multi-processors," in *International Conference on Parallel Architectures and Compilation Techniques*, 2013.

[26] S. Pagani, H. Khdr, W. Munawar, J.-J. Chen, M. Shafique, M. Li, and J. Henkel, "TSP: Thermal Safe Power: Efficient power budgeting for many-core systems in dark silicon," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, 2014.

[27] H. Khdr, T. Ebi, M. Shafique, H. Amrouch, and J. Henkel, "mdtm: multi-objective dynamic thermal management for on-chip systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2014, p. 330.

[28] J. Cui and D. Maskell, "A fast high-level event-driven thermal estimator for dynamic thermal aware scheduling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 6, pp. 904–917, 2012.

[29] G. Dhiman and T. Rosing, "System-level power management using online learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 5, pp. 676–689, 2009.

[30] H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran, "Energy-efficient adaptive pipelined mpsocs for multimedia applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 5, pp. 663–676, 2014.

[31] R. Ye and Q. Xu, "Learning-based power management for multicore processors via idle period manipulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 7, pp. 1043–1055, 2014.

[32] U. A. Khan and B. Rinner, "Online learning of timeout policies for dynamic power management," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 4, pp. 96:1–96:25, 2014.

Anup Das Dr. Anup Das received the B.Eng. degree in Electronics and Telecommunication Engineering from Jadavpur University, India, in 2004. He received the Ph.D. degree in computer engineering in the area of embedded systems from the National University of Singapore, in 2014. He is currently a post-doctoral research fellow at the University of Southampton. His research interests include thermal and energy-aware computing.

Geoff V. Merrett Geoff V. Merrett (GSM06M09) received the BEng degree (Hons) in Electronic Engineering and the PhD degree from the University of Southampton, UK, in 2004 and 2009 respectively. He is currently an Associate Professor in energy-efficient electronic systems at the University of Southampton. He has research interests in energy-efficient embedded systems and low-power pervasive computing.

Mirco Tribastone Mirco Tribastone is Associate Professor of Computer Science at IMT Institute for Advanced Studies, Lucca. Previously, he has held positions at the University of Southampton and at the Ludwig-Maximilians University of Munich. He received his Ph.D. from Edinburgh University in 2010. His main interests are in the quantitative evaluation of systems, and abstraction and model reduction techniques with applications to software performance engineering and computational biology.

Bashir M. Al-Hashimi Bashir M. Al-Hashimi (M99-SM01-F09) is the Dean of the Faculty of Physical Sciences and Engineering at University of Southampton, UK. He is ARM Professor of Computer Engineering and Co-Director of the ARM-ECS research centre. His research interests include low-power design and test of embedded computing systems.

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

$$N_{TC}(i) = A_{TC}\,(\delta T_i - T_{Th})^{-b}\, e^{\frac{E_a}{KT_{max}(i)}} \tag{21}$$

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.

$$MTTF = \frac{N_{TC}\sum_{i=1}^{m} t_i}{m} \tag{22}$$

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

$$N_{TC} = \frac{m}{\sum_{i=1}^{m} \frac{1}{N_{TC}(i)}} \tag{23}$$

Combining Equations 21-23, $MTTF = \frac{A_{TC}\sum_{i=1}^{m} t_i}{\textit{Thermal Stress}}$, 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.

$$\textit{Thermal Stress} = \sum_{i=1}^{m} (\delta T_i - T_{Th})^b \times e^{\frac{-E_a}{KT_{max}(i)}} \tag{24}$$

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

APPENDIX B
ALTERNATIVE CHANGE-POINT DETECTION TECHNIQUES

Difference-Based Change Point Detection

In this technique, the difference between previous and current workload segment's CPU cycles count is compared with a predefined threshold; a change is inferred if this difference is greater than the threshold. The storage requirement for this technique is $2 * N_{Bits}$, where N_{Bits} is the number of bits required to store the CPU cycles count of a workload segment. System designers typically use a set of known workloads to determine this threshold, such that all change points in workload (including the unknown ones) can be detected at run-time. However, as we show in Section V, difference-based detection has a lower detection accuracy (being sensitive to the choice of this detection threshold) for real-time data, resulting in performance and power penalties. To select the threshold for all applications, a system designer needs to know all workload that will be executed on the system apriori. The alternative approach is to characterize the threshold using training workloads. Training with known workloads does not guarantee to detect all changes as we demonstrate in Section V.

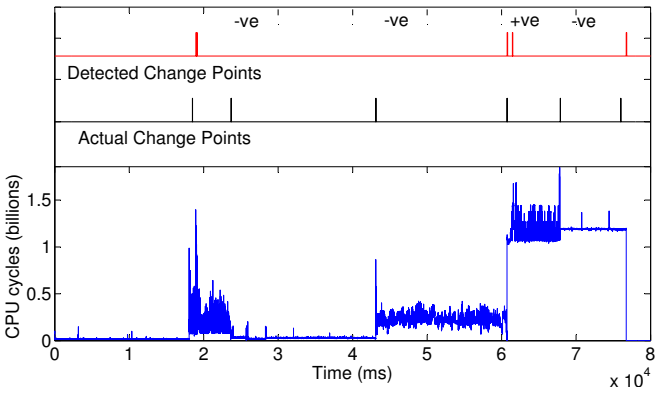

Fig. 16. Manhattan distance-based autonomous change point detection for MPEG4 application decoding a 1080p video. CPU cycles per video frame is identified in blue; actual change points are identified in black; and the Manhattan distance-based change points are identified in red.

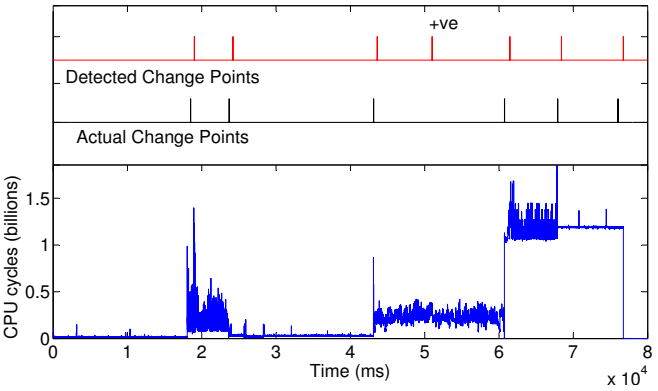

Fig. 17. Statistical distance-based autonomous change point detection for MPEG4 application decoding a 1080p video. CPU cycles per video frame is identified in blue; actual change points are identified in black; and the Statistical distance-based change points are identified in red.

Geometric Distance-Based Change Point Detection

Let W_{test} and W_{ref} denote the test and the reference window, respectively. Let the function $d()$ indicate the dissimilarity (also called the **distance**) between these windows. The null and the alternate hypothesis are defined as

$$\begin{aligned} \texttt{Null Hypothesis:} \quad & \mathcal{H}_0 : d(W_{test}, W_{ref}) \leq \delta \\ \texttt{Alternate Hypothesis:} \quad & \mathcal{H}_1 : d(W_{test}, W_{ref}) > \delta \end{aligned}$$

where δ is the threshold used to decide whether a change has occurred. Let $X_1^{ref}, \cdots X_N^{ref}$ be the N samples of the the reference window and $X_1^{test}, \cdots X_N^{test}$ be that of the test window. These two windows are projected to a N-dimensional space and represented as points. The reference and the test points have co-ordinates $(X_1^{ref}, \cdots X_N^{ref})$ and $(X_1^{test}, \cdots X_N^{test})$, respectively. The dissimilarity between the two windows therefore translates to distance between the two points in the N-dimensional space. Several distance measurement approaches have been proposed in literature. Examples include Manhattan, Euclidean and Chebychev distances. These distance functions are defined as

$$d(W_{test}, W_{ref}) = \sum_{i=1}^{N} |X_i^{test} - X_i^{ref}| \qquad \texttt{Manhattan}$$

$$d(W_{test}, W_{ref}) = \sqrt{\sum_{i=1}^{N} \left(X_i^{test} - X_i^{ref}\right)^2} \qquad \texttt{Euclidean}$$

$$d(W_{test}, W_{ref}) = \max_{i=1}^{N} |X_i^{test} - X_i^{ref}| \qquad \texttt{Chebychev}$$

These geometric distances are usually good at detecting abrupt changes in data. However, gradual changes remain undetected in most cases. Additionally, the threshold δ is dependent on workload, and therefore this approach has a similar limitation as the difference-based detection.

Geometric Distance vs Statistical Distance

Figure 16 plots CPU cycles (blue) obtained from the MPEG4 application decoding a 1080p video. These CPU cycles are collected at every frame interval and correspond to cycles consumed in decoding a video frame. As seen from this figure, CPU cycles change within application execution at time instances 18s, 24s, 44s, 60s, 68s and 77s with corresponding changes in video resolution. These change points are identified in black. Change points detected using the Manhattan distance-based autonomous technique is shown in red. Compared to the actual ones, the Manhattan distance-based technique detects four change points – three real and one false positive. This approach also has three false negative change points. Detection quality of this approach is 0.33. The Euclidean and the Chebychev distance-based detection techniques have detection qualities similar to the Manhattan distance-based one.

Statistical distance-based detection results are shown in Figure 17 for the same workload. As seen from this figure, this approach detects seven change points – six real and one false positive. Detection quality using this approach is 0.83, an improvement of 2.5x compared to the geometric distance-based detection. This high detection quality motivated us to use statistical distance for change point detection.