# Learning Transfer-based Adaptive Energy Minimization in Embedded Systems

Rishad A. Shafik, *Member, IEEE*, Sheng Yang, *Member, IEEE*, Anup Das, *Member, IEEE*, Luis A. Maeda-Nunez, *Member, IEEE*, Geoff V. Merrett, *Member, IEEE* & Bashir M. Al-Hashimi, *Fellow, IEEE*

***Abstract*—Embedded systems execute applications with varying performance requirements. These applications exercise the hardware differently depending on the computation task, generating varying workloads with time. Energy minimization with such workload and performance variations within (intra) and across (inter) applications is particularly challenging. To address this challenge we propose an online approach, capable of minimizing energy through adaptation to these variations. At the core of this approach is a reinforcement learning algorithm that suitably selects the appropriate voltage/frequency scaling (VFS) based on workload predictions to meet the applications' performance requirements. The adaptation is then facilitated and expedited through learning transfer, which uses the interaction between the application, runtime and hardware layers to adjust the VFS. The proposed approach is implemented as a power governor in Linux and extensively validated on an ARM Cortex-A8 running different benchmark applications. We show that with intra- and inter-application variations, our proposed approach can effectively minimize energy consumption by up to 33% compared to the existing approaches. Scaling the approach to multi-core systems, we also demonstrate that it can minimize energy by up to 18% with 2X reduction in the learning time when compared with an existing approach.**

***Index Terms*—Energy-efficiency, dynamic voltage/frequency scaling, reinforcement learning.**

## I. Introduction

Energy minimization is a prime design objective for embedded systems. To enable energy minimization these systems are equipped with processors with dynamic voltage and frequency scaling (DVFS) capabilities, controlled by the system firmware; examples include Linux's power governors and ARM's VFS firmware [1]. The basic principle of DVFS is to reduce the operating voltage/frequency (V/F) dynamically at runtime, resulting in a cubic decrease in power consumption [2], [3], [4].

Energy minimization approach through DVFS can be broadly classified into two types – offline and online. The offline approach characterizes the workloads of a given application exploiting application-specific knowledge. The profiled workloads are then used during runtime to adjust the power control levers at regular intervals for achieving energy minimization. Workload characterization and energy minimization of video decoders [5], [6] and control-theoretic formulation of energy consumption of multimedia workloads [7] are typical examples.

Online energy minimization approach has the basic principle of controlling the hardware power levers based on the processor workloads [8], [9]. Depending on the control mechanism, online approach can be reactive or proactive. In the reactive approach the VFS is controlled based on the history of CPU workloads. When the CPU workload is higher/lower than a pre-defined value, an increased/decreased V/F is used, e.g. Linux's ondemand power governor [11]. In the proactive approach predicted workloads are used to manage the hardware power control levers [10], [12]. The impact of such control is then observed and adjusted through feedback from the hardware performance monitors. Jung *et al.* [13] proposed one such approach using an initial value problem based processor workload classification. The workloads are then predicted and classified to continuously determine V/F for energy minimization. Ramakrishna *et al* [14] showed a similar online approach for task workload classification and VFS control with the feedback from the performance counters.

Since processor workloads are exercised differently depending on the application tasks, Siyu *et al.* [15] and Shen *et al.* [17] have proposed online approaches using machine learning algorithm. Their approaches have shown methods learning the VFS required for an application to achieve energy minimization in the presence of performance variations due to application-generated CPU workloads. However, these approaches do not consider the variation of application performances, such as frame rate for video decoders, page loading rate for browsers etc. Among others, learning-based idle-time manipulation has been proposed in [18] to reduce energy in multi-core systems.

Modern embedded systems feature workload and performance variations both within and across applications [16]. As these variations arise dynamically due to the types of computation being executed, energy minimization using the existing approaches is challenging (see Section II). This is because the online approaches, such as [11], [14], [17] do not interact with the applications for their changing performance needs, which leads to either over-performance or failure in meeting their performance requirements. Moreover, existing approaches using machine learning [15], [17], [18] use a single runtime formulation of V/F scaling for a given performance requirement, which cannot adapt to intra- and inter-application variations.

S. Yang, A. Das, L.A. Maeda-Nunez, G.V. Merrett and B.M. Al-Hashimi are with the School of ECS, University of Southampton, UK e-mail: {sy2u12,akd1g13,lm15g10,gvm,bmah}@ecs.soton.ac.uk.

R.A. Shafik is affiliated with the School of EEE, Newcastle University, UK e-mail: rishad.shafik@newcastle.ac.uk.

Manuscript first received on 18 Dec. 2014, revised on 4 April and 14 July, accepted on 20 August 2015.

To effectively minimize energy consumption in the presence of such workload and performance variations, this paper makes the following specific ***contributions***:

- an energy minimization approach to effectively adapt to the intra- and inter-application variations is proposed,
- fundamental to the approach is a reinforcement learning (RL) algorithm to suitably control VFS for a given performance requirement, followed by a learning transfer (LT) algorithm to adapt to variations, and
- a Linux runtime governor implementation of the approach is shown for extensive validation using different applications.

To the best of authors' knowledge, this is the first work that shows learning transfer-based adaptive energy minimization and its implementation on single and multi-core embedded systems. The remainder of this paper is organized as follows. Section II motivates the proposed approach, Sections III describes the approach and its implementation. Sections IV and V reports the experimental results and overheads analysis, Section VI demonstrates scaling of the approach to multi-core systems. Finally, Section VII concludes the paper.

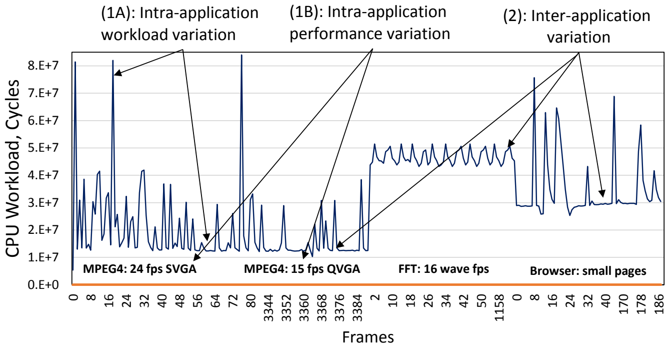


Fig. 1. Workload and performance variations within and across applications

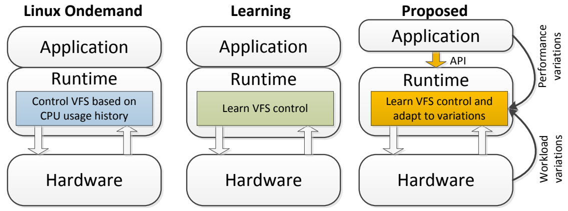
## II. Motivation

Fig. 1 shows an example of workload (in CPU cycles) and performance variations (in ms per frame) within (intra) and across (inter) applications, considering three scenarios: MPEG4 [20] followed by FFT [21] and browser (based on [22]) rendering small html pages (less than 20kb in size) from *bbench* [23]. The CPU workloads were recorded on a DM3730 SoC from Texas Instruments, integrated on the BeagleBoard-xM (BBxM) platform [19], which incorporates an ARM Cortex-A8 CPU core running at 800MHz. The following two observations are made:

***Observation 1***: The CPU workloads and performances vary within an application. For example, MPEG4 decoding at 24 SVGA frames per second (fps) exhibits up to 7x workload variation, Fig. 1 (1A). Such variations arise due to decoding of intra-coded (I) SVGA frames with higher computations, followed by a number of predictive-coded (P) frames with lower computations [27]. The MPEG4 also experiences a performance change from 24 SVGA fps to 15 QVGA fps, Fig. 1 (1B).

***Observation 2***: The CPU workload profiles change when the system switches between applications. For example, when the system switches from MPEG4 to FFT, the workload increases by 3x on average (Fig. 1(2)), since FFT wave frames are computationally intensive. Also, when the system switches from FFT to browser, the workload decreases by 2x due to less computations required. During such switches the performance requirements also change from 67 ms per MPEG4 frame to 62 ms per FFT wave frame and then to 100 ms per browser page.

Minimizing energy consumption in the presence of the above-mentioned variations can be particularly challenging as the processor power control levers will need to be continually learnt and adjusted. This paper proposes a learning transfer-based adaptive energy minimization approach, addressing such learning and adaptation aspects. To highlight the importance of such approach, Fig. 2 shows three energy minimization approaches applied to the application scenarios in Fig. 1: MPEG4 decoding at 24 SVGA fps (41 ms/frame) and 15 QVGA fps (i.e. 67 ms/frame), FFT processing 16 wave fps (i.e. 62 ms/frame) and browser rendering at 100 ms/page. The energy consumption incurred by the applications are measured using an Agilent DC Power Analyzer (N6705B) (see Section IV for details).



| Requirement | Energy | Avg. Performance | Energy | Avg. Performance | Energy | Avg. Performance |
|---|---|---|---|---|---|---|
| SVGA: 41 ms/frame | 229 J | 35 ms/frame | 189 J | 37 ms/frame | 178 J | 40 ms/frame |
| QVGA: 67 ms/frame | 108 J | 53 ms/frame | 126 J | 40 ms/frame | 97 J | 66 ms/frame |
| FFT: 62 ms/frame | 164 J | 50 ms/frame | 123 J | 67 ms/frame | 131 J | 61 ms/frame |
| Browser: 100 ms/page | 119 J | 79 ms/frame | 89 J | 112 ms/page | 101 J | 98 ms/page |
| | (a) | | (b) | | (c) | |

Fig. 2. Comparison between energy minimization approaches: (a) Linux's ondemand [11], (b) learning-based [17], and (c) proposed

The learning approach (Fig. 2.(b)) carries out a single formulation of the VFS controls based on the predicted workloads using machine learning [17]. However, as the VFS controls resulting from such learning approach do not adapt to workload and performance variations across the applications, it cannot achieve effective energy minimization. For example, after initially learning VFS controls for the MPEG4 decoding at 24 fps, the learning approach over-performs and incurs higher energy consumption for the MPEG4 decoding 15 QVGA fps. Conversely, in the case of FFT and browser applications it under-performs and violates the specified performance requirements (highlighted in red). Due to performance-agnostic nature of VFS controls the Linux ondemand over-performs for most of the applications, incurring higher energy consumptions (Fig. 2.(a)). The proposed approach provides the lowest energy consumption for all intra- and inter-application variations. This is because it learns the appropriate VFS controls and adapts to performance and workload variations across all applications (Fig. 2.(c)).

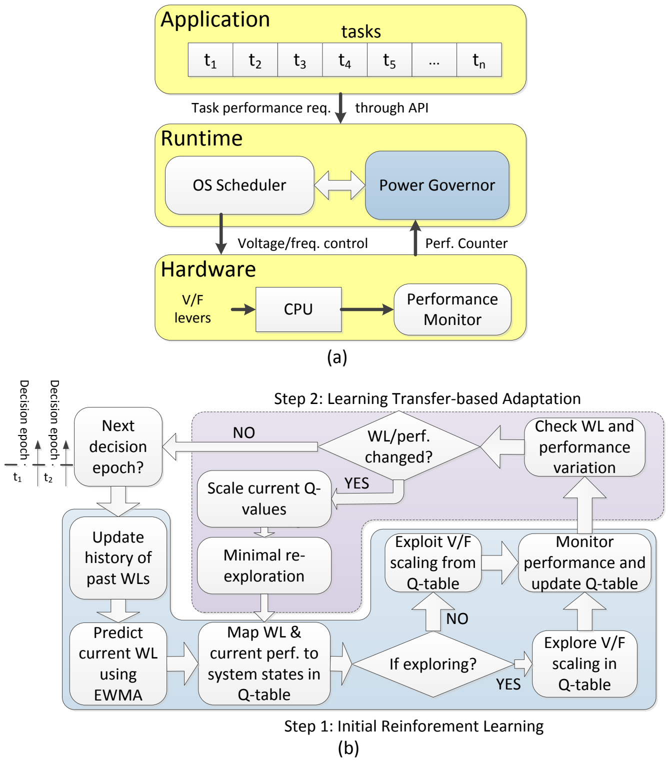


Fig. 3. (a) Proposed adaptive approach showing interactions between layers, (b) flowchart of the proposed adaptive energy minimization approach

## III. Proposed Energy Minimization Approach

Fig. 3.(a) shows the proposed approach highlighting the interactions between the application, runtime and hardware layers. The application layer consists of a series of tasks being executed at time intervals (referred to as decision epochs). Each task has a performance requirement, specified through an application programming interface (API) as

```
rts.set_perf(41);
```

where *rts* is a thread-safe runtime variable (see Appendix A), *set_perf* sets the performance requirement as 41 ms per decision epoch. The runtime layer consists of the power governor implementing the proposed approach. With a given performance requirement, the governor minimizes energy through suitably controlling the hardware power levers (i.e. VFS) at regular decision epochs.

A flowchart of the proposed approach is depicted in Fig. 3.(b), showing two major steps: reinforcement learning (RL) and learning transfer (LT) . The RL sets up proactive VFS controls at each decision epoch through state prediction. When performance or workload variations are detected, these controls are adapted through a LT algorithm. These steps are detailed in the following.

### *A. Step 1: Reinforcement Learning*

The initial learning through RL algorithm evolves in three phases, as follows.

*1) State Prediction and Q-table Formation:* State prediction is a required phase in the RL step to identify the Q-value of the future system state. In our approach, the same predictor is also used to classify the expected workload to a system state at the beginning of each decision epoch, similar to [14], [15], [17]. This predictor estimates the current CPU workload based on the history of the past workloads and maps the workload to a system state based on the current performance. The CPU cycles' count is preferred as the workload parameter over the other parameters, such as memory accesses, cache misses and instruction rate, etc. as it directly defines the CPU activity when executing instructions of a task. To predict the workload, an exponential weighted moving average (EWMA) scheme is used, similar to [5], [8]. Using this scheme, the predicted workload for the $t+1^{th}$ decision epoch, $\hat{\mathcal{C}_{t+1}}$ is

$$\hat{\mathcal{C}_{t+1}} = \omega \mathcal{C}_t + \sum_{i=t-2}^{t-D} (1-\omega)^i \, \mathcal{C}_i \quad , \qquad (1)$$

where $\mathcal{C}_t$ and $\mathcal{C}_i$ are the previous observed workloads (in CPU cycles) at the $t^{th}$ and $i^{th}$ decision epochs, $(t-D) \leq i \leq (t-1)$, $\omega$ is the moving average coefficient and $D$ is the window size of past observed workloads. The $\omega$ and $D$ values are chosen to give higher prediction accuracy for the given application workloads, similar to [14]. However, the workload prediction through (1) still undergoes mispredictions during runtime due to variations in workloads. The impact of such mispredictions on the corresponding VFS controls is discussed in Section IV-A.

The system state is determined by using the $\hat{\mathcal{C}}$ through (1) and the current performance as a pair. The state space $\mathcal{S}$ is comprised of the combinations of average slack ratios ($\mathcal{L}$) and $\hat{\mathcal{C}}$, denoted as $\mathcal{S}\{\mathcal{C}, \mathcal{L}\}$. For each state ($s_t$), the average slack ratio at $t_{th}$ decision epoch ($\mathcal{L}_t$) is divided in bins (five, for example) as

$$\forall_{s_t}: \begin{cases} \mathcal{L}_t > 0.15, \;\; \mathcal{L}_t \leq -0.15, \\ 0.05 < \mathcal{L}_t \leq 0.15, \;\; -0.15 < \mathcal{L}_t \leq -0.05, \;\; \& \\ |\mathcal{L}_t| \leq 0.05 \end{cases} \qquad (2)$$

Similarly, for each state CPU workloads are divided in several workload bins (six, for example) as:

$$\forall_{s_t}: \begin{cases} \mathcal{C}_t > [\mathcal{C}_b + 2\Delta\mathcal{C}]\,; \;\; higher\ bin, \\ (\mathcal{C}_b + m\Delta\mathcal{C}) > \mathcal{C}_t \geq [\mathcal{C}_b + (m-1)\Delta\mathcal{C}]\,; m = -1:2, \\ \mathcal{C}_t < [\mathcal{C}_b - 2\Delta\mathcal{C}]\,; \;\; lower\ bin. \end{cases} \qquad (3)$$

where $\Delta\mathcal{C}$ is the size of the workload bins around a base workload ($\mathcal{C}_b$). With the given combinations between $\mathcal{L}_t$ and $\hat{\mathcal{C}}_{t+1}$ in (2) and (3), state entries for the Q-table are set up. Thus, for each predicted workload ($\hat{\mathcal{C}_{t+1}}$) and the current performance ($\mathcal{L}_t$) pair the system state is mapped using (2) and (3).

The state space, and the action space (formed of the VFS control options, denoted as $\mathcal{A}\{Vdd, f\}$) define the size of Q-table for the RL step. The size of the Q-table in terms of the total number of state-action pairs ($|\mathcal{A}\{Vdd, f\}| \times |\mathcal{S}\{\mathcal{C}, \mathcal{L}\}|$) is important for the RL algorithm as it influences the trade-off between learning overhead and energy minimization achieved (see Section III-B and IV for detailed trade-offs). In this work, the size of Q-table is carefully chosen to ensure a good trade-off between learning overhead and energy minimization (see Section V). With the given state prediction and Q-table formation, the RL algorithm carries out exploration and evaluation of the VFS control actions as discussed next.

*2) Exploration:* Exploration is a crucial phase in RL algorithm as does the actual learning of appropriate VFS actions based on the system states. Traditionally, it is carried out using a random action selection strategy from the pool of actions, each with a uniform probability distribution (UPD). However, such exploration is inefficient as it does not use the intuitive relationships that often exist between the state-action pairs [24]. To reflect such relationship during exploration, we use the following discrete exponential probability distribution (EPD) function for the selection of action ($a_t$)

$$p(a_t)_{a_t \in \mathcal{A}\{Vdd,f\}} = \lambda \exp\left[-\lambda f(a) \beta \mathcal{L}\right], \tag{4}$$

where $\lambda$ is the uniform probability of actions (i.e. $\lambda = 1/|\mathcal{A}\{Vdd,f\}|$), $f(a)$ is the operating frequency in action $a$ and $\beta$ is a constant. According to (4) when $\mathcal{L}$ is close to zero, the exploration probabilities are almost uniform, guided by $\lambda$. However, positive and negative $\mathcal{L}$ prioritize selection of lower and higher frequencies, respectively. The probability distribution, given by (4), has an advantage in terms of quicker learning, which can be stated and proven by Lemma I (see Appendix B). At the beginning of the $(t+1)^{th}$ decision epoch, the Q-value corresponding to a selected VFS action is updated as [15]:

$$Q(s_{t+1}, a_t) = Q(s_t, a_t)(1-\alpha) + \alpha[r_t + \gamma \max_{a_t} Q(s_{t+1}, a_t)], \tag{5}$$

where $\alpha$ is the learning rate, $\gamma$ is the discount factor to de-scale the current maximum Q-value in the row ($0 \leq \alpha, \gamma \leq 1$), $s_t$ is the observed state in the $t^{th}$ decision epoch and $s_{t+1}$ is the predicted state in the $(t+1)^{th}$ decision epoch determined by estimated workload and current performance (see Section III-A1). The reward function $r_t$ in (5) is computed as a function of the resulting average slack ratio at the $t^{th}$ decision epoch ($\mathcal{L}_t$) and it's change since the last decision epoch ($\Delta\mathcal{L}$), i.e.

$$r_t = K_1|\mathcal{L}_t| + K_2 \Delta\mathcal{L} \quad , \tag{6}$$

where $K_1$ and $K_2$ denote constant values, pre-determined to ensure actions that improve $\mathcal{L}_t$ values (through $\Delta\mathcal{L}$ trends) are rewarded or vice versa. The $\Delta\mathcal{L}$ values are estimated by

$$\mathcal{L}_t = \frac{1}{N(T_{ref})} \sum_{i=0}^{t} \left(T_{ref} - T_i - T_{OVH}\right), \tag{7}$$

where $T_{ref}$ is the reference execution time, $T_i$ is the application task execution time, $N$ is the number of decision epochs elapsed since application started with a given $T_{ref}$, $T_{OVH}$ is the total overheads caused by learning and adaptation steps (see Section V) and $\Delta\mathcal{L}_t$ is the average slack difference since the last observation, given as $\Delta\mathcal{L}_t = \mathcal{L}_{t-1} - \mathcal{L}_t$. The $T_i$ in (7) can be estimated as the ratio between the observed processor CPU cycles ($C_i$) and the operating frequency chosen ($f_i$) at $i^{th}$ decision epoch as

$$T_i = \frac{C_i}{f_i}. \tag{8}$$

Equations (5) and (6) set up the exploration of VFS control actions.

*3) Exploitation:* In this phase, the state-action relationships learnt are exploited. The transition from the exploration to exploitation phase is controlled through the exploration probability (EP), denoted by $\epsilon$ ($0 \leq \epsilon \leq 1$). To accelerate exploitation $\epsilon_t$ at the $t^{th}$ decision epoch is updated as

$$\epsilon_t = \epsilon_{t-1} \exp\left[-(1-\alpha)\right], \tag{9}$$

where $\alpha$ is the learning factor per decision epoch. Based on the $\epsilon_t$ value, the exploration or exploitation is carried out to find the best policy sub-set ($\pi^*(s_t, a_t)$) from a set of exploration policies ($\Pi(s_t, a_t)$, $\pi \in \Pi$) as follows:

$$\pi^*(s_t, a_t) = \begin{cases} a_t : \max\left(Q(s_t, a)\right); & \text{if } p > \epsilon_t, \\ a_k : p(a_k) \text{ is given by } (4) \end{cases} \tag{10}$$

where $p$ is a random value uniformly distributed over $[0,1]$. As can be seen, when $\epsilon_t$ value decreases in (9), the probability of exploitation increases in (10).

TABLE I
THE COMPARISONS BETWEEN SINGLE Q-TABLE [9] AND LEARNING TRANSFER APPROACHES

| $\lvert\mathcal{A}\rvert$ | $\Delta C$ | Q-table size: $\lvert\mathcal{A}\rvert \times \lvert\mathcal{S}\rvert$ (single Q-table) | Q-table size: $\lvert\mathcal{A}\rvert \times \lvert\mathcal{S}\rvert$ (learning transfer) |
|---|---|---|---|
| 4 | $1\times10^7$ | 4000 | 120 |
| 4 | $2\times10^7$ | 2000 | 120 |
| 4 | $4\times10^7$ | 1000 | 120 |
| 6 | $1\times10^7$ | 6000 | 180 |
| 6 | $2\times10^7$ | 3000 | 180 |
| 6 | $4\times10^7$ | 1500 | 180 |

### *B. Step 2: Learning Transfer-based Adaptation*

Using a single Q-table in RL algorithm step for covering the dynamic ranges of workload and performance variations can expand the learning space substantially. Table I shows example illustrations of the impact of using a single Q-table approach, similar to [9]; column 1 and 2 show the number of actions and the size of workload bins, while column 3 shows the number of state-actions pairs considering a workload coverage from 0 to $10^{10}$ cycles. As can be seen, for such a dynamic workload a single Q-table will have 4000 state-action pairs considering 4 actions with workload bin size of $10^7$ each. The size of the Q-table can expand further to 6000 for 6 actions.

To ensure a quicker learning and adaptation to dynamic workload or performance variations, smaller Q-table is used in this work together with LT. Columns 5 and 6 show the motivation of using such LT-based adaptation (Table I). As expected with a smaller state space of 30, the number of state-action pairs is 120 for 4 actions, and 180 for 6 actions around a base workload. With smaller Q-tables the LT also benefits from quicker convergence and learning of the Q-table. Table II compares the worst-case convergence times between single Q-table approach and smaller Q-tables in the LT-based approach considering both workload and performance variations in the case of $\Delta C$=$10^7$ and $|\mathcal{A}|$=4. Columns 1-3 show the number of workload variations and the corresponding number of decision epochs required for full convergence of both approaches. As can be seen, the single Q-table approach takes invariably 4000 decision epochs despite any workload variations for the full learning of the Q-table. The LT-based approach takes significantly lower number of decision epochs for the same due to smaller Q-tables. The convergence time in this approach, however, depends on the number of workload variations. As can be seen, when the number of workload variation increases from 1 to 8, the the convergence time increases from 160 to 440 decision epochs, which is still significantly lower than the single Q-table approach.

TABLE II
COMPRATIVE CONVERGENCES OF SINGLE Q-TABLE APPROACH [9] AND PROPOSED LEARNING TRANSFER-BASED APPROACH

| Workload variation | | | Performance variation | | |
|---|---|---|---|---|---|
| No. of Variations | Learning convergence | | No. of Variations | Learning convergence | |
| | single | transfer | | single | transfer |
| 0 | 4000 | 120 | 0 | 4000 | 120 |
| 1 | 4000 | 160 | 1 | 8000 | 160 |
| 2 | 4000 | 200 | 2 | 12000 | 200 |
| 4 | 4000 | 280 | 4 | 20000 | 280 |
| 8 | 4000 | 440 | 8 | 36000 | 440 |

Table II also compares the worst case convergence times of both approaches for different performance variations (columns 4-6). As can be seen, the single Q-table approach requires significantly higher number of decision epochs when performance variations (i.e. change in $T_{ref}$) are encountered. This is because the original Q-table can no longer provide the the optimized VFS scaling options with such variation, necessitating re-learning from scratch. Unlike the single Q-table approach, the LT can continue to exploit the previous learning and converge faster to give the optimized VFS options for the new system states (Lemma I compares and proves the convergences times, see Appendix B).

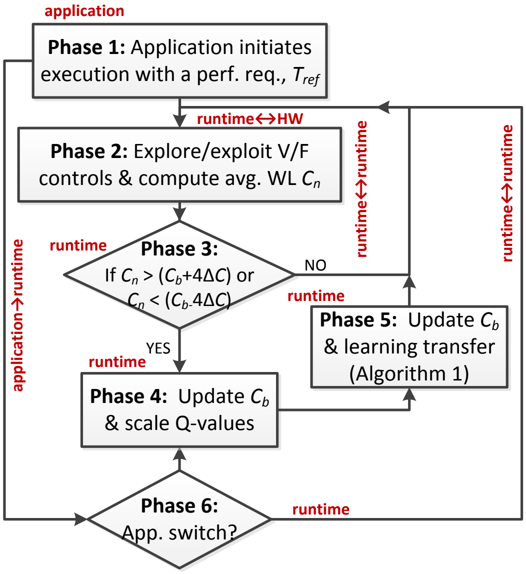

application
Phase 1: Application initiates execution with a perf. req., $T_{ref}$
runtime↔HW
Phase 2: Explore/exploit V/F controls & compute avg. WL $C_n$
runtime
Phase 3: If $C_n > (C_b+4\Delta C)$ or $C_n < (C_b-4\Delta C)$
NO
YES
runtime
runtime
Phase 4: Update $C_b$ & scale Q-values
Phase 5: Update $C_b$ & learning transfer (Algorithm 1)
Phase 6: App. switch?
runtime
application→runtime
runtime↔runtime
runtime↔runtime

Fig. 4. Learning transfer-based adaptation to workload/performance variations

When the workload profile changes within an application beyond the current base workload (observation 1), the current Q-table fails to minimize energy consumption effectively. The same is also true when the performance requirement changes within and across applications (observation 2). To enable adaptation to these variations the proposed approach first detects these variations through inter-layer interactions (Fig. 4). This is then followed by a Q-table update through LT algorithm (Algorithm 1). The detection of workload and performance variations and their adaptations are further detailed next.

*1) Adaptation to Workload Variation:* The workload variation is detected through the interaction between the runtime and the hardware layers in the following phases, as shown in Fig. 4. Initially, with a given performance requirement of $T_{ref}$ per application task (Fig. 3.(a)), the runtime learns VFS control actions (phases 1-2). During this time the runtime also computes the short-term average workload, $\mathcal{C}_n$, over the last $n$ decision epochs (the impact of varying $n$ is studied in Section V). The mean workload, $\mathcal{C}_n$, is then compared with the current base workload ($\mathcal{C}_b$) in the Q-table (phase 3). If $\mathcal{C}_n$ is confirmed as beyond the current table limits (i.e. $\mathcal{C}_n > (\mathcal{C}_b + 4\Delta\mathcal{C})$ or $\mathcal{C}_n < (\mathcal{C}_b - 4\Delta\mathcal{C})$), the Q-table states are updated and scaled with a new base workload ($\mathcal{C}_b$) nearest to $\mathcal{C}_n$ (phase 4). The scaling from the old Q-table is carried out as follows:

$$\forall_t : Q(s_t, a_t)_{scaled} = Q(s_t, a_t)_{old} \exp\left[-\frac{1}{\mathcal{L}\rho_{WL}}\right], \quad (11)$$

where $\rho_{WL}$ is the scaling ratio proportional to $\frac{\mathcal{C}_{b_{new}}}{\mathcal{C}_{b_{old}}}$. Note that the Q-value scaling in (11) ensures that the Q-values are downscaled according to the $\mathcal{L}$ states. At higher $\mathcal{L}$ values the $Q(s_t, a_t)_{scaled}$ values are less downscaled, while at lower $\mathcal{L}$ values the $Q(s_t, a_t)_{scaled}$ values are more downscaled to ensure that further exploration of VFS control actions can update the optimal policy $\pi^*$ quickly. To minimize such exploration, the current best policy ($\pi^*(s_t, a_t)$) in the scaled Q-table is then updated in the next phase using LT algorithm, as shown in Algorithm 1 (phase 5).

**Algorithm 1** Learning transfer algorithm

```
Require: $\rho_{WL}$, $Q(s_t,a_t)_{scaled}$, $\pi^*(s_t,a_t)_{old}$
 1: for each $s_t$ in $Q(s_t,a_t)_{new}$ do
 2:     if $s_t$ is explored then
 3:         if $\mathcal{L}(s_t)$ is near-zero (i.e. $\pm5\%$) then
 4:             set: $f(\pi'(s_t,a_t)_{new}) = \rho_{WL} \times f(\pi^*(s_t,a_t)_{old})$
 5:         else
 6:             set: $f(\pi'(s_t,a_t)_{new}) = f(\pi^*(s_t,a_t)_{old})$
 7:         end if
 8:         map: $f(\pi'(s_t,a_t)_{new})$ to the nearest action $a'_t$ in $Q(s_t,a_t)_{new}$
 9:         for every action in $Q(s_t,a_t)_{new}$ do
10:             if action is $a'_t$ then
11:                 swap $Q(s_t,a'_t)_{new}$ with $Q(s_t,a^*_t)_{scaled}$
12:             else
13:                 set: $Q(s_t,a_t)_{new} = \alpha Q(s_t,a_t)_{scaled}$
14:             end if
15:         end for
16:     else
17:         set: $Q(s_t,a_t)_{new} = Q(s_t,a_t)_{scaled}$
18:     end if
19: end for
20: return $\pi'(s_t,a_t)_{new}$ and $Q(s_t,a_t)_{new}$
```

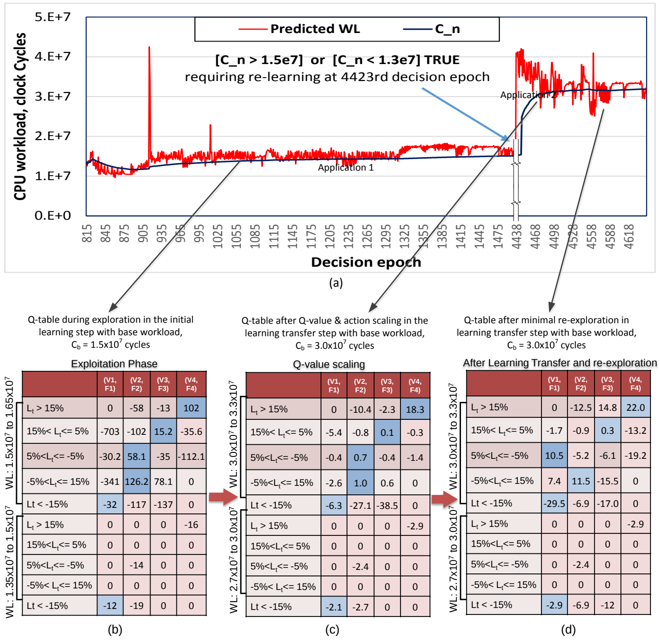
As can be seen, for each already explored state ($s_t$) the chosen frequency in the new policy ($f(\pi'(s_t, a_t)_{new})$) is obtained through scaling by a factor of $\rho_{WL}$ (lines 2-6). For a state with near-zero slack (i.e. $\pm5\%$), such scaling requires multiplying the old chosen frequency by the scaling ratio

($\rho_{WL}$), while for a state with higher positive or negative slack the old chosen frequency is retained as the new chosen frequency. After such scaling of chosen frequencies, their corresponding actions are mapped in the new Q-table and the new Q-values are updated through transfer of the scaled Q-values (lines 9-15). The Q-value of the new chosen action is set as the Q-value of the old chosen action (line 11), while the other values retained from the scaled Q-values through (11). For un-explored or partially explored state ($s_t$), the scaled Q-value is retained in the new Q-table (line 17). The resulting Q-table (line 20) with transferred learning is then used with a reduced exploration probability ($\epsilon_t$) for accelerated re-exploration, instead of learning from the scratch.

*2) Adaptation to Performance Variation:* When the application performance requirement changes due to intra- or inter-application switch, the adaptation is enabled through the API-based interaction between application and runtime layer (phase 6, Fig. 4). Upon such interaction, the runtime layer learns the new $T_{ref}$ with the old $\mathcal{C}_b$ and carries out Q-value scaling and LT (phases 4 and 5). Similar to (11), the Q-value scaling is carried out as

$$\forall_t : Q(s_t, a_t)_{scaled} = Q(s_t, a_t)_{old} \exp\left[-\frac{1}{\mathcal{L}\rho_T}\right], \qquad (12)$$

where $\rho_T$ is the scaling ratio, proportional to $\frac{T_{ref_{old}}}{T_{ref_{new}}}$. Similar to (11), Equation (12) also scales $Q(s_t, a_t)_{old}$ values based on the $\mathcal{L}$ values. Following the Q-value scaling in (12), the Q-values are transferred between action pairs using the Algorithm 1 with $p_{WL}$ values replaced by $p_T$. The transferred Q-values are then used for further minimal explorations (Fig. 3.(b)). The LT-based adaptation has the advantage of lower number of re-explorations required when compared with the re-learning based approach. The reduction of the number of explorations is described through Proposition I (see Appendix B).



(a)

Q-table during exploration in the initial learning step with base workload, $C_b$ = 1.5x10$^7$ cycles

Exploitation Phase

| | | (V1, F1) | (V2, F2) | (V3, F3) | (V4, F4) |
|---|---|---|---|---|---|
| WL: 1.5x10$^7$ to 1.65x10$^7$ | $L_t$ > 15% | 0 | -58 | -13 | 102 |
| | 15%< $L_t$<= 5% | -703 | -102 | 15.2 | -35.6 |
| | 5%<$L_t$<= -5% | -30.2 | 58.1 | -35 | -112.1 |
| | -5%<$L_t$<= 15% | -341 | 126.2 | 78.1 | 0 |
| | Lt < -15% | -32 | -117 | -137 | 0 |
| WL: 1.35x10$^7$ to 1.5x10$^7$ | $L_t$ > 15% | 0 | 0 | 0 | -16 |
| | 15%<$L_t$<= 5% | 0 | 0 | 0 | 0 |
| | 5%<$L_t$<= -5% | 0 | -14 | 0 | 0 |
| | -5%< $L_t$<= 15% | 0 | 0 | 0 | 0 |
| | Lt < -15% | -12 | -19 | 0 | 0 |

(b)

Q-table after Q-value & action scaling in the learning transfer step with base workload, $C_b$ = 3.0x10$^7$ cycles

Q-value scaling

| | | (V1, F1) | (V2, F2) | (V3, F3) | (V4, F4) |
|---|---|---|---|---|---|
| WL: 3.0x10$^7$ to 3.3x10$^7$ | $L_t$ > 15% | 0 | -10.4 | -2.3 | 18.3 |
| | 15%< $L_t$<= 5% | -5.4 | -0.8 | 0.1 | -0.3 |
| | 5%<$L_t$<= -5% | -0.4 | 0.7 | -0.4 | -1.4 |
| | -5%<$L_t$<= 15% | -2.6 | 1.0 | 0.6 | 0 |
| | Lt < -15% | -6.3 | -27.1 | -38.5 | 0 |
| WL: 2.7x10$^7$ to 3.0x10$^7$ | $L_t$ > 15% | 0 | 0 | 0 | -2.9 |
| | 15%<$L_t$<= 5% | 0 | 0 | 0 | 0 |
| | 5%<$L_t$<= -5% | 0 | -2.4 | 0 | 0 |
| | -5%< $L_t$<= 15% | 0 | 0 | 0 | 0 |
| | Lt < -15% | -2.1 | -2.7 | 0 | 0 |

(c)

Q-table after minimal re-exploration in learning transfer step with base workload, $C_b$ = 3.0x10$^7$ cycles

After Learning Transfer and re-exploration

| | | (V1, F1) | (V2, F2) | (V3, F3) | (V4, F4) |
|---|---|---|---|---|---|
| WL: 3.0x10$^7$ to 3.3x10$^7$ | $L_t$ > 15% | 0 | -12.5 | 14.8 | 22.0 |
| | 15%< $L_t$<= 5% | -1.7 | -0.9 | 0.3 | -13.2 |
| | 5%<$L_t$<= -5% | 10.5 | -5.2 | -6.1 | -19.2 |
| | -5%<$L_t$<= 15% | 7.4 | 11.5 | -15.5 | 0 |
| | Lt < -15% | -29.5 | -6.9 | -17.0 | 0 |
| WL: 2.7x10$^7$ to 3.0x10$^7$ | $L_t$ > 15% | 0 | 0 | 0 | -2.9 |
| | 15%<$L_t$<= 5% | 0 | 0 | 0 | 0 |
| | 5%<$L_t$<= -5% | 0 | -2.4 | 0 | 0 |
| | -5%< $L_t$<= 15% | 0 | 0 | 0 | 0 |
| | Lt < -15% | -2.9 | -6.9 | -12 | 0 |

(d)

Fig. 5. Example illustration of learning transfer-based adaptation

*3) Example Illustration:* To illustrate how the proposed approach adapts to inter-application switch (from Application 1 to Application 2) as an example, Fig. 5.(a) plots the predicted ($\hat{\mathcal{C}}_t$) and observed mean workload ($\mathcal{C}_n$) values, with $n$=300 decision epochs, while Fig. 5.(b)-(d) show Q-tables for ten states covering two workload bins around the mean workload and five slack state variations as shown in (eq:statedef1) out of total 30 states (for demonstration purposes) and four actions for both applications. As can be seen in Fig. 5.(b), after the initial learning and exploration the Q-table is populated with two different kinds of states: explored states and partially explored or un-explored states. The explored states are more frequently visited due to workloads exercised by the hardware due to Application 1. As a result, most of the actions are evaluated and the best actions are chosen as the highest Q-value in the table (highlighted in blue). The un-explored or partially explored states are not visited as often and hence not all actions are evaluated (un-explored actions are marked by zero values). After initial exploration and learning of the controls, the Q-table facilitates exploitation for the Application 1 with a $\mathcal{C}_b$=1.5 × 10$^7$.

After the 4423rd decision epochs the system switches from Application 1 to Application 2 with both workload and performance variations. The workload variation is observed through the mean workload $\mathcal{C}_n$, while performance variation is directly communicated by the application to the runtime (Fig. 4). To adapt to such variations, the proposed approach carries out LT in two stages. First, the Q-values are scaled through (11) using the new base $\mathcal{C}_b$=3.0×10$^7$ near the current mean workload ($C_n$). The resulting scaled Q-values are shown in Fig. 5.(c). The scaled Q-values are then used to carry out further action transfer of explored and un-explored states through Algorithm 1. For explored states, the new actions near zero values (±5%) are further minimally re-explored. The resulting Q-values and the chosen actions after such exploration are shown in Fig. 5.(d). It is to be noted that LT requires only 1 out of 10 states to be explored compared all 10 states in other approaches.

## IV. Experimental Results

The proposed adaptive energy minimization approach is implemented as a power governor in Linux kernel revision 3.7.10 (see Appendix A) running on DM3730 SoC, integrated on the BeagleBoard-xM (BBxM) platform [19]. The platform consists of, among others, a single-core ARM Cortex-A8 CPU core, which supports four V/F levels: 300MHz at 0.93V, 600MHz at 1.10V, 800MHz at 1.26V and 1GHz at 1.35V [19]. To evaluate the effectiveness of the proposed governor, ffmpeg-based multimedia [20], MiBench benchmark [21] and browser [22] applications are executed. The energy consumptions of the ARM Cortex-A8 core are measured through direct observation of current and voltage using an Agilent DC Power Analyzer (N6705B). The current was observed by lifting an inductor off of the board and re-routing the signal through the same inductor and the power analyzer, while the voltage supplied across the Cortex-A8 was measured directly across the core supply. All experiments are carried out using a Q-table size of (30x4) consisting of 5 slack states and 6 workload states as such table size gives the best trade-off between energy minimization and learning overheads as discussed in Section V.

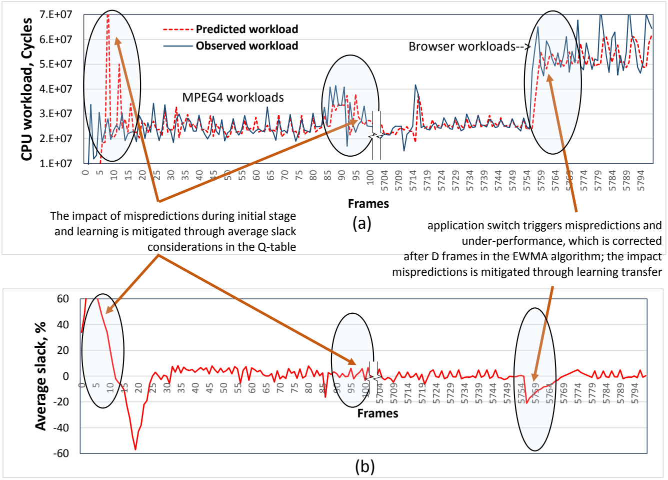


Fig. 6. (a) CPU workload predictions of MPEG4, followed by FFT, (b) impact of learning on the average slack ratio (in %)

### A. Impact of State Prediction and Exploration

To investigate into the impact of state prediction on learning, Fig. 6.(a) shows the predicted and observed workloads (in CPU cycles), while Fig. 6.(b) shows the corresponding average slack ratios ($\mathcal{L}$) caused by the RL algorithm (Section III-A) for MPEG4 decoding at 24 SVGA fps, followed by browser application. As can be seen, the EWMA based workload prediction (given by (1)) incurs occasional mispredictions: during the exploration (the first 25 frames) and exploitation phases (after 90 frames) of MPEG4 and also briefly when the system switches from MPEG4 to browser (after 5750 frames, Fig. 6.(a)). The highest average misprediction of about 8% on average with respect to the mean observed workload was observed during the initial 100 frames in the MPEG4 decoding 24 SVGA fps, while the lowest misprediction of only 3% was observed for the following frames. To mitigate the impact of such mispredictions the RL algorithm (Section III-A) considers the current performance offset in terms of average slack ratio ($\mathcal{L}_t$) together with the currently predicted workload during state mapping of the next state. In the event of performance offset (showing positive or negative high $\mathcal{L}_t$) caused by mispredictions, the RL algorithm learns and applies the appropriate VFS controls to minimize it through the action rewarding mechanism (see Section III-A2). Similar to MPEG4, workload misprediction is also observed when the system switches to the FFT application after the $5754^{th}$ frame. At this time the LT and further explorations take place, which causes under-performance initially for about 45 FFT frames. However, after further explorations during the next 50 frames, the under-performance is offset by updating the learning of the appropriate VFS controls.

To highlight the advantages of the explorations using EPD during the initial learning step (i.e. RL step), Table III compares the average number of explorations required by the proposed approach and the that of the learing-based approach [17]. Column 1 shows the applications with the input sizes used, while Columns 2 and 3 show the number of explorations recorded for the proposed and learning-based approaches. As can be seen, the proposed approach benefits from reduced number of explorations due to the relationship between current performance and VFS action in (4) when compared with the exploration using a UPD in [17] (see Lemma I, Appendix B). The applications FFT and susan were found to have the lowest number of explorations as these applications exhibit less workload variation, i.e. less number of workload states during the learning step. As a result, the RL algorithm learns the VFS controls faster. The MPEG4, H.264 and browser applications showed the highest number of explorations due to higher workload variations and eventually higher number of workload states visited during learning.

TABLE III
COMPARATIVE NUMBER OF EXPLORATIONS DURING THE INITIAL LEARNING IN THE PROPOSED AND LEARNING APPROACHES

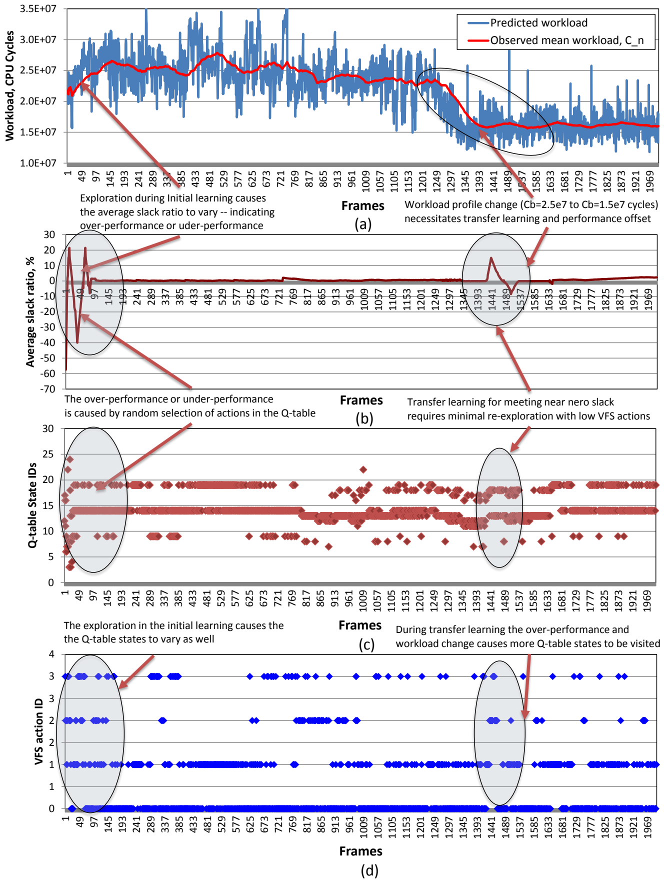
| Application | No. explorations (proposed) | No. explorations (learning [17]) |
|---|---|---|
| MPEG4 (30 fps) | 81 | 144 |
| H.264 (15 fps) | 91 | 149 |
| mad (22k) | 86 | 149 |
| susan (384x288) | 78 | 135 |
| ispell (largespell) | 82 | 139 |
| FFT (32 fps) | 75 | 119 |
| browser (small) | 89 | 141 |

### B. Intra-Application Energy Minimization

A number of experiments are carried out with intra-application workload and performance variations showing comparative evaluation of the proposed adaptive approach.

*1) Workload Variations:* Fig. 7 shows the experimental results of an H.264 decoder decoding at 24 VGA fps, used as a case study, highlighting the adaptation to intra-application workload variations. Fig. 7.(a) shows the predicted workload together with the observed mean CPU workload over a moving window of $n$=200 decision epochs, while Fig. 7.(b)-(d) show the resulting average slack ratios caused by RL algorithm and adaptation steps, the Q-table states and the VFS control actions chosen over the decision epochs (in terms of frames). As can be seen, initially the governor starts to learn the VFS control actions (Fig. 7.(d)), which results in performance offset in terms of $\mathcal{L}$ (Fig. 7.(b)). As the governor initiates exploiting some of these VFS controls, $\mathcal{L}$ starts to reduce. During this phase the Q-table states vary depending on the predicted workloads as the VFS actions are chosen from the Q-table (Fig. 7.(c)-(d)).

After the $1271^{th}$ frame the workload profile changes within the application (observation 1, Section II), which is detected through comparison of the observed mean workload, $C_n$ with the base workload ($\mathcal{C}_b$) of the Q-table (Fig. 4). Due to such variation in the workload, the proposed governor carries out LT from the old Q-table with $\mathcal{C}_b$=$2.5 \times 10^7$ to the new Q-table with $\mathcal{C}_b$=$1.5 \times 10^7$ (Section III-B1). The LT is then followed by further exploration of the Q-table to update the VFS controls to achieve near-zero $\mathcal{L}$ values. Due to such re-exploration,



Exploration during Initial learning causes the average slack ratio to vary -- indicating over-performance or uder-performance

Workload profile change (Cb=2.5e7 to Cb=1.5e7 cycles) necessitates transfer learning and performance offset

The over-performance or under-performance is caused by random selection of actions in the Q-table

Transfer learning for meeting near nero slack requires minimal re-exploration with low VFS actions

The exploration in the initial learning causes the Q-table states to vary as well

During transfer learning the over-performance and workload change causes more Q-table states to be visited

Fig. 7. (a) Predicted and observed mean workloads, (b) average slack ratios (in %), (c) VFS controls, and (d) corresponding Q-table states of an H.264 decoder

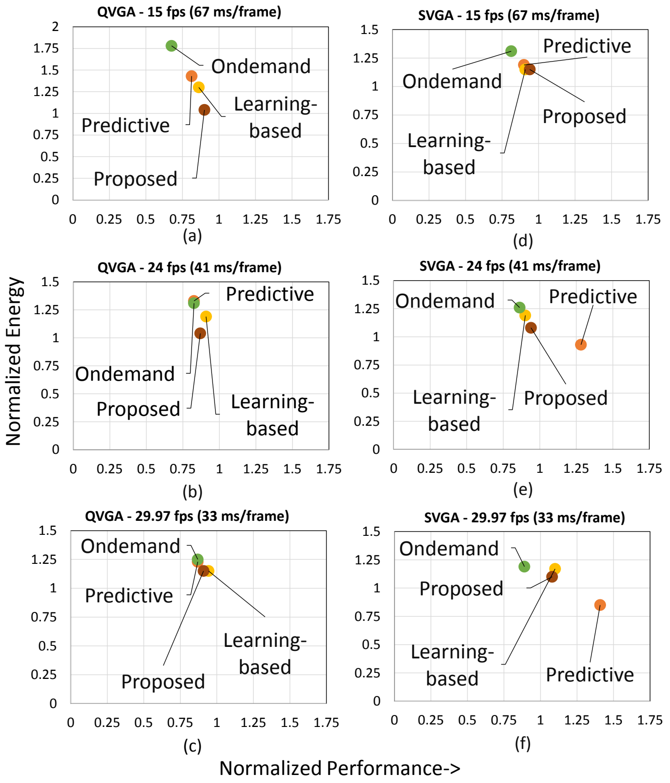


Fig. 8. Comparative evaluation of energy minimization of H.264 video decoder with different performance requirements (normalized performance of 1 means on par performance, $>$ 1 means under-performance and $<$ 1 means over-performance; normalized energy of $>$ 1 means higher energy consumption, $<$ 1 means lower energy consumption; $\approx$1 means effective energy minimization provided that performance is also on par)

the $\mathcal{L}$ values are perturbed again with over-performance at reduced $C_n$ (Fig. 7.(b)). Since the proposed governor adapts the VFS controls in the presence of intra-application workload variations, it can effectively minimize energy, while meeting the application performance requirement.

Fig. 8 plots the normalized energy and performance values of the proposed energy minimization approach with different frame rates and resolutions of H.264, compared with the existing approaches. Normalization is carried out to give comparative figures between different approaches covering the dynamic range of performance and workload variations for various applications. Figures 8.(a)-(c) show the results of decoding QVGA resolution with 15 fps, 24 fps and 29.97 fps, while Figures 8.(d)-(f) show the same for decoding SVGA resolutions. The performance is normalized with respect to the required performance per frame ($T_{ref}$) and the energy normalization is carried out with respect to Oracle, found through offline determination of optimized VFS controls for the observed CPU workloads. The normalized energy and performance results are obtained through averaging the decoder results of three different video clips (*football*, *flower* and *foreman*) from *xiph* video repository (http://media.xiph.org/video/derf/). The results of the proposed approach are compared with Linux's ondemand governor [11], predictive and learning-based approaches. Predictive approach is implemented using [14] without any explicit performance information from the application.

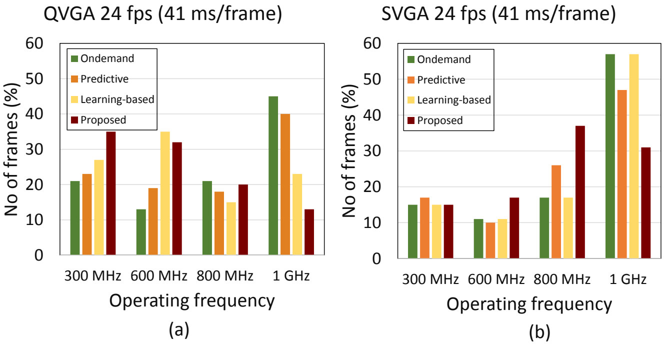


Fig. 9. Comparative histogram of operating frequencies applied in H.264 decoding (a) QVGA, and (b) SVGA frame resolutions, 24 fps each

As can be seen, the ondemand governor consistently over-performs when compared with the other approaches as it is agnostic of application performance requirements. Hence, it generates the highest energy consumption among all approaches. The predictive energy minimization approach is also oblivious to applications' performance requirement, and hence it fails to minimize energy consumption effectively meeting the applications' performance requirements. For example, in the case of QVGA the predictive approach over-performs and incurs higher energy consumption (Fig. 8(a)-(c)). However, for

SVGA with 24 and 29.97 fps it under-performs, which makes the energy savings achieved through the predictive approach ineffective (Fig. 8(e)-(f)). The learning-based approach [17] performs better as it learns the VFS controls based on the performance requirements. However, since it uses a single Q-table formulation it adapts poorly to intra-application workload variations. Our proposed approach can adapt to such variations and minimize energy effectively. However, energy reduction achieved in our approach depends on the number of intra-application workload variations detected. For example, in the case of Fig. 8.(c)-(f), the proposed approach does not offer much of an energy saving when compared to the learning-based approach [17] due to less workload variations (1, in the case of decoding SVGA frames at 24 fps). However, in the case of Fig. 8.(a)-(b), up to 20% energy reduction can be achieved as the number of workload variations are much higher (4 for both cases: decoding QVGA frames at 15 and 24 fps).

To give further insight into the energy minimization of different approaches compared (Fig. 8), Fig. 9 plots the histograms of H.264 decoding QVGA and SVGA resolutions at 24 fps each. As can be seen from Fig. 9.(a), for the low resolution QVGA video, the learning-based, predictive and proposed approaches execute most of the frames at 300MHz or 600MHz. The ondemand, however, executes more than 45% of the frames at 1 GHz due to its CPU utilization-based VFS control. For decoding videos with different workloads (Fig. 9.(b)), the proposed approach generates a balanced frequency utilization based on the performance requirement to ensure effective energy minimization.

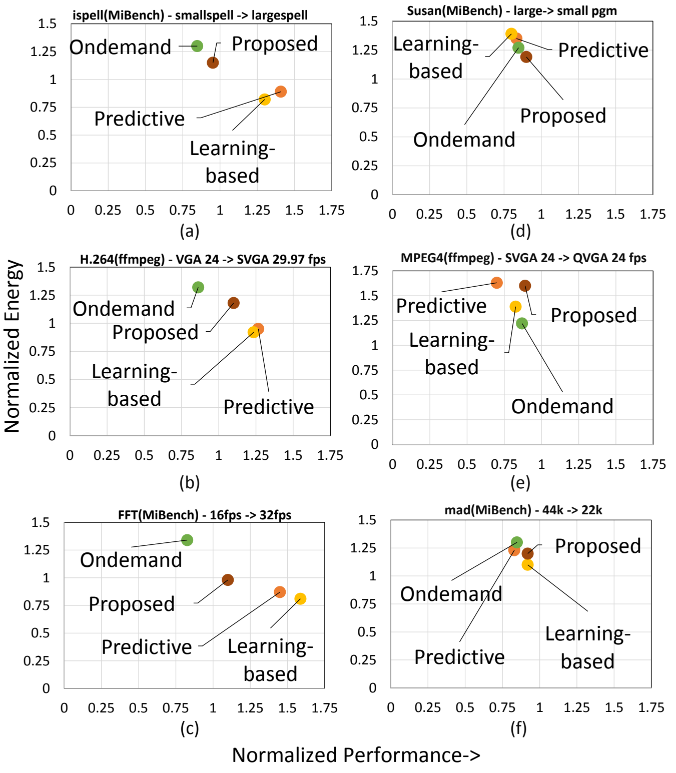


Fig. 10. Comparative energy and performance trade-offs with intra-application performance variations for different benchmark applications (performance and energy consumption values normalized similar to Fig. 8)

2) *Performance Variations:* Fig. 10 depicts the normalized energy and performance values (with respect to Oracle) of different benchmark applications with varied performances within the applications. Figures 10.(a)-(c) show the variation from a lower performance to a higher performance for ispell (MiBench), H.264 (ffmpeg) and FFT (MiBench) applications, while Figures 10.(d)-(f) show the variation from a higher performance to a lower performance for susan (MiBench), MPEG4 (ffmpeg) and mad (MiBench) applications. The normalized energy and performance results are obtained through averaging three observations, each with input sequence of 3000 decision epochs (i.e. frames for MPEG4, H.264, FFT and susan, spelling task for ispell and audio packet for mad).

As can be seen, when the performance requirement increases from low to high, both predictive and learning-based approaches under-perform (Fig. 10.(a)-(c)). On the other hand, when the performance requirement decreases, these approaches over-perform and incur higher energy consumptions (Fig. 10.(d)-(f)). This is because both approaches cannot adapt to performance variations due to lack of interactions between the layers. The ondemand governor, however, shows trends of scaling with the performance requirements, but it consistently over-performs. The proposed approach can effectively scale with the variation in the performance requirement and adapt through LT (Section III-B2). As a result, it shows effective energy minimization across all experiments saving on average 33% and 24% when compared with the predictive and learning-based approaches in the case of MPEG4 performance variation (Fig. 10.(e)) and 30% when compared with the ondemand governor in the case of FFT performance variation (Fig. 10.(c)).

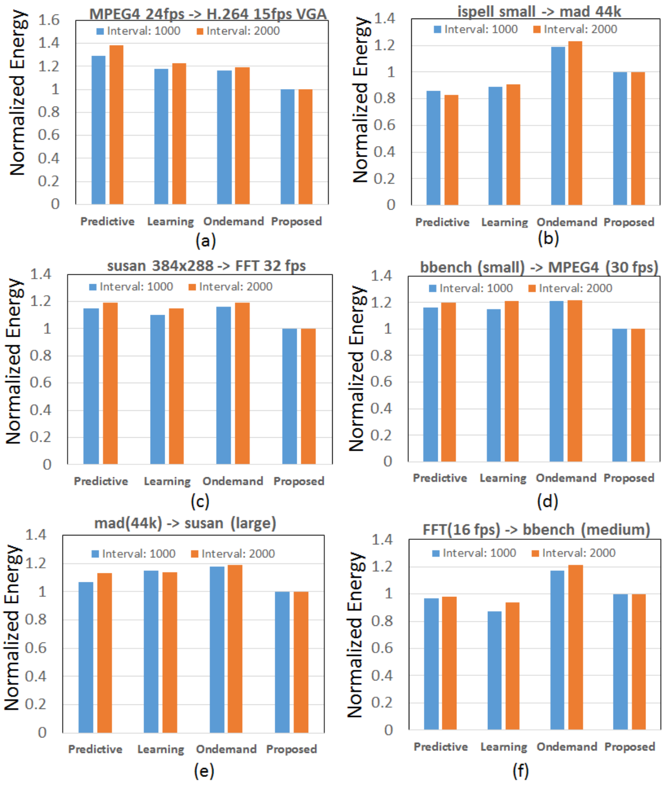


Fig. 11. Comparative normalized energy with inter-application variations

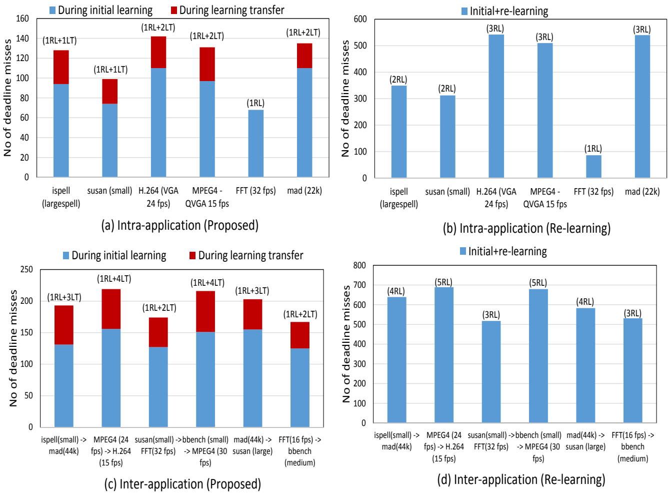
### C. Inter-Application Energy Minimization

Fig. 11 plots the comparative normalized energy consumptions of energy minimization approaches for six inter-application scenarios. Fig. 11.(a) and (b) show inter-application switches from high performance to low performance, Fig. 11.(c) and (e) show the same for high performance to high performance switch, and finally, Fig. 11.(d) and (f) show inter-application switches from low performance to high performance. For each inter-application variation, two different switching intervals of 1000 and 2000 decision epochs are used. The energy values are normalized with respect to the proposed approach.

From Fig. 11 two observations can be made. First observation is related to inter-application energy minimization for a given switching interval; as can be seen, for an application switch from a higher performance to lower performance requirement the proposed approach saves up to 38% and 22% for switching interval of 2000 decision epochs compared to the predictive and learning-based approaches, respectively (Fig. 11.(a)). However, when the application switches from a lower to higher performance requirement the proposed approach consumes more energy, while meeting the new application performance requirement when compared with the predictive and learning-based approaches. This is because the proposed adaptive approach adapts to higher VFS control actions to meet the increased performance requirement. Both predictive and learning-based approaches fail to meet the application performance requirement despite their energy savings. Similar adaptations are also observed for the other inter-application switches. As expected, the ondemand consistently over-performs compared to the proposed approach. The second observation is related to the change of switching interval; as can be seen with higher switching interval the proposed approach exploits the learning and adaptation for longer time. This leads to higher energy savings compared with the other approaches. For example, for a change of switching interval from 1000 to 2000, the proposed approach saves up to 6% more energy compared to the predictive approach.

## V. Learning Overheads

The learning algorithims in the proposed approach have the following two impacts: deadline misses and learning overheads. To demonstrate the impact of learning through real-time deadline misses and overhead, Fig. 12.(a)-(b) show the worst-case number of deadline misses for intra-application scenarios, while Fig. 12.(c)-(d) show the same for inter-application scenarios using the proposed and learning [17] approaches, respectively. For all application scenarios, the worst-case number of deadline misses was recorded from five consecutive runs using the input sizes specified, each application with 3000 decision epochs. For the intra-application scenarios (Fig. 12.(a)-(b)), the following observation can be made. As can be seen, the number of worst-case deadline misses depends on the intra-application workload variations and the initial random explorations. The mad, MPEG4 and H.264 applications exhibit the highest number of deadline misses as these applications go through one initial RL and two intermediate LTs each. For the inter-application scenarios (Fig. 12.(c)-(d)), the worst-case number of deadline misses depends on the number of variations encountered. Hence, the inter-application scenarios with MPEG4 (24 fps) to H.264 (15 fps) and browser (small) to MPEG4 (30 fps) incur higher number of deadline misses, as they go through one initial RL and four LTs. These deadline misses accounts to only 4.8% (compared to 18% in [17]) for the intra-application scenarios and 3.8% (compared to 13% in [17]) for the inter-application scenarios over 3000 decision epochs.



(a) Intra-application (Proposed)

(b) Intra-application (Re-learning)

(c) Inter-application (Proposed)

(d) Inter-application (Re-learning)

Fig. 12. Worst-case number of deadline misses for (a) intra-application variations using the proposed approach, (b) intra-application variations using the learning approach [17], (c) inter-application variations using the proposed approach, and (d) inter-application variations using the learning approach [17]

To demonstrate the impact of learning due to additional computation and storage, Fig. 13 plots the average learning overheads of the proposed approach, compared with the existing approaches: ondemand [11], learning [17] and predictive [14]. The measured time overheads are evaluated by averaging the differences of per frame execution times of a *ffmpeg* video decoder decoding three sequences ($T_{ref} = 31ms$) with and without energy minimization approaches. The overheads have the following three components:

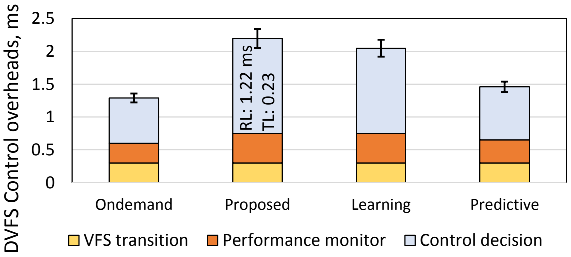


Fig. 13. Comparative time overheads ($T_{OVH}$) of different approaches

1. *VFS transition delay* is a variable delay due to transition of CPU frequencies. ARM Cortex A8 has a transition delay of about 300 $\mu$s [30].
2. *Performance monitor delay* includes the time taken by clock (depends on the number of accesses) and performance counter register access (typically $\approx$20$\mu$s per access using C-wrapped

assembly instructions).

3. *Control Decision Delay* includes the time taken by the control steps and varies/depends on their complexities.

As expected, the control decision dominates the overheads as it requires number of computation steps (such as learning and transfer algorithms with fixed point calculations). The proposed approach exhibits the highest time overhead of 2.1ms, with up to 8% deviation due to random explorations during initial RL and intermediate LTs. Compared with the learning approach [17], the proposed approach also uses additional interactions between the layers and LT-based adaptation steps to minimize energy further in the presence of variations, with up to 1.22 ms and 0.23 ms, respectively, for the RL and TL steps. The predictive and ondemand approaches have lower overheads due to simpler control decisions and less performance counters' access. The higher overhead of the proposed approach highlights one of the limitations of the proposed approach, which can be minimized by defining the decision epochs as multiple of frame intervals.

To demonstrate the impact of learning choices made in terms of size of the Q-tables in the RL algorithm, Fig. 14 plots the average energy (in %) and time overheads (in ms) for different Q-table sizes with the following state-action entries: 15 states and 4 actions (i.e. 15x4), 30 states and 4 actions (i.e. 30x4) and 40 states and 4 actions (i.e. 40x4). The energy overheads are evaluated by comparing the average energy consumptions of the proposed approach with offline profile-generated energy consumption in Oracle for similar performance levels (Fig. 14.(a)), while time overheads are measured by observing the CPU times elapsed during learning and VFS action ($T_{OVH}$) averaged per decision epoch (Fig. 14.(b)). Both measurements are obtained through two ffmpeg (H.264 and MPEG4) and four MiBench (FFT, susan, ispell and mad) benchmark applications, running over 3000 decision epochs each with the corresponding input sequences. As can be seen, the energy overheads increase slightly with the increased Q-table sizes (Fig. 14.(a)) due to the following two reasons. Firstly, with higher number of states, the Q-tables now have higher complexity and require longer exploration times, which results in slower convergence over time. Secondly, due to increased time overheads, the effective deadline per decision epoch ($T_{ref} - TOVH$) reduces, which requires slightly higher VFS control to be applied to meet the performance requirements, resulting in higher energy consumption.

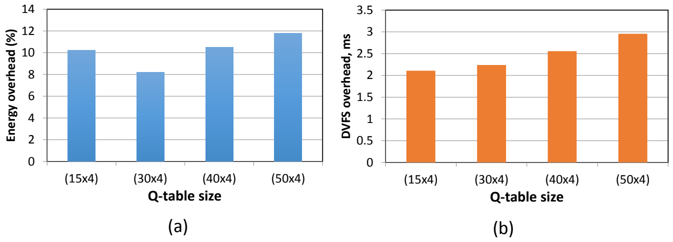


Fig. 14. (a) Energy, and (b) time overheads for different Q-table sizes

To investigate into the impact workload bin ($\Delta C$) and window sizes of average workload ($C_n$) computation, Fig. 15(a) and (b) show the 3D bar plots of the energy and time overheads incurred by the proposed approach for the following $\Delta C$ values: $0.05 \times C_b$, $0.1 \times C_b$, $0.2 \times C_b$ and $0.25 \times C_b$. For each $\Delta C$ value, the window size $n$ is varied between 100, 200, 300 and 1000 decision epochs. The energy and time overheads are evaluated repeating the experiments reported in Fig. 14 with the Q-table size of (30x4). Fig. 15(a)-(b) demonstrate the energy consumption and time overhead trade-offs with $\Delta C$ values for a given moving average window size. As can be seen, at the lower $\Delta C$ values, the energy and time overheads are higher as the workload variations covered by the Q-table is lower. As a result more workload variations are encountered, which incur the higher LT overheads, as expected. When the $\Delta C$ values are higher, coarser workload variations are covered by the Q-table, which gradually reduce the learning time overheads due to less number of LTs. However, at increased $\Delta C$ values, the normalized energy overhead increases marginally due to loss of precision of VFS controls.

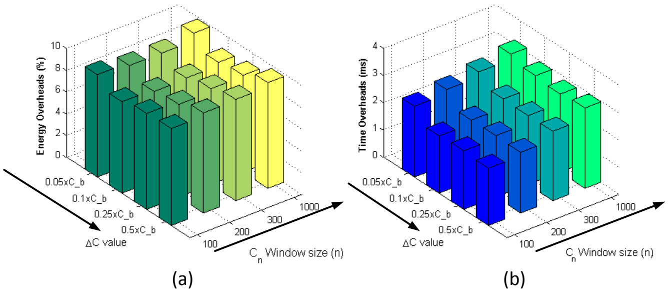


Fig. 15. (a) Energy, and (b) time overheads for varying workload bins ($\Delta C$)

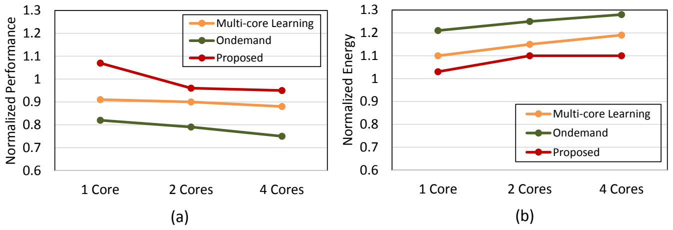
The moving average window size for $C_n$ also demonstrates energy and time overheads trade-offs (Fig. 15(a)-(b)). At lower window size, the proposed approach experiences higher workload variations in the short-term, causing an increase in the LTs needed to adapt to workload variations. This causes the energy and time overheads to increase slightly. As the window size is gradually increased, the number of intra-application variations decrease, reducing the energy and time overheads. However, when the window size is too large, it causes higher energy and time overheads due to loss of control precisionand higher computation and storage overheads. Such increased time overheads, in turn, reduces the opportunity to reduce energy effectively as demonstrated by the highest energy overheads ($\approx$ 10%) for a given $\Delta C$ value of $0.05 \times C_b$ (Fig. 15(a)). The best trade-off between energy and time overheads is obtained at $\Delta C$ value=$0.1 \times C_b$ and $C_n$ window size of 200.

## VI. Scaling to Multi-core Systems

The proposed approach is also implemented and validated in multi-core systems, through simple modifications to the approach in Section III. First, the predicted workload per core is normalized with respect to the total system workload as

$$\overline{C}^j_{t+1} = \frac{\hat{C}^j_{t+1}}{\sum_j^J \hat{C}^j_{t+1}} \quad , \tag{13}$$

where $\hat{C}^j_{t+1}$ is the predicted workload in CPU cycles and $\overline{C}^j_{t+1}$ is the normalized workload for the $j$-th processor core ($j$=1 to $J$, $J$ is the total number of cores in the system). With the given normalized workload ($\overline{\mathcal{C}}^j_{t+1}$) and the average slack ratio ($\mathcal{L}_t$) bins a number of Q-table states are defined and organized in rows (similar to (2) and (3)). For each state, the available VFS control options are used in the action space organized in the columns to form the Q-table. This Q-table is then shared among the processor cores to allow RL through one core action update per decision epoch (controlled in a round robin fashion). Such VFS control per decision epoch has two distinct advantages: (a) automatic LT between cores when similar workload is predicted (due to normalization of workload in (13)), and (b) Q-table complexity is reduced significantly as opposed to controlling multiple cores per decision epoch, which requires combinations of VFS controls of all cores in the Q-table [29].



(a)

(b)

Fig. 16. Comparative (a) normalized performance, and (b) normalized energy consumptions of different approaches for varying number of processor cores

To validate the effectiveness of the approach in multi-core systems, further experiments are carried out on the following: a Xilinx Zync ZC702 SoC [26] with two ARM Cortex-A9 cores, and a Hardekernel Odroid-XU SoC [28] with four ARM cores. The Zync SoC supports three VFS control points: 666MHz at 1V, 333MHz at 0.8V and 222MHz at 0.75V and the Odroid-XU SoC has six VFS control points: 1.6GHz at 1.2V, 1.0GHz at 1V, 600MHz at 0.8V and 300MHz at .75V. An H.264 based video decoder application is executed with a *football* sequence of approximately 3000 frames using the following three approaches: multi-core DVFS control approach [29], Linux ondemand governor per core [11] and the proposed approach. The approach [29] was chosen for comparison as it is the closest match using RL-based DVFS controls in multiprocessor systems. However, for equivalence and comparability between [29] and our approach the thermal constraint was removed. Fig. 16.(a) and (b) show the normalized performance and energy consumptions of the approaches. Similar to Fig. 8, the performance is normalized with respect to $T_{ref}$ and the energy normalization of the Oracle.

As can be seen, the proposed approach continues to provide energy reduction compared to the existing approaches. The multi-core learning [29] and ondemand [11] approaches overperform due to poor adaptation to variations, resulting in up to 18% higher energy (Fig. 16.(b)). Using LT-based approach to multi-core systems has the added advantage of controlled learning overheads as shown in Fig. 17. As the learning of each core can be exploited by the other cores, it shows quicker convergence.

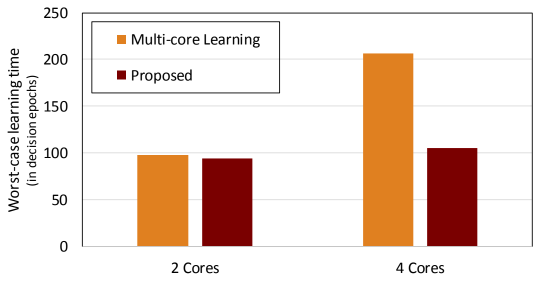


Fig. 17. Comparative worst-case learning overheads for multi-core systems

## VII. Conclusions

An adaptive energy minimization approach for embedded systems has been proposed, capable of adjusting to workload and performance variations within and across applications. The energy minimization is enabled through RL algorithm for identifying the suitable VFS controls based on predicted workloads for a given application performance requirement. To ensure VFS controls are adjusted to workload or performance variations learning transfer-based adaptation is carried out, guided by the feedback from the CPU performance counters. The proposed approach is implemented as a power governor in Linux OS and extensively validated through experiments using different benchmark applications and number of cores. The approach is expected to provide effective energy savings for embedded systems that typically execute multiple applications.

## Acknowledgment

The authors would like to thank the EPSRC-UK for funding this work under PRiME project, grant number EP/K034448/1. Experimental data used in this paper can be found at DOI http://dx.doi.org/10.5258/SOTON/383899.

## References

[1] D. Flynn. An ARM Perspective on Addressing Low-power Energy-efficient SoC Designs. in *Proc. of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'12)*, pp.73–78, 2012.

[2] Y. Tan, W. Liu, and Q. Qiu. Adaptive power management using reinforcement learning. in *Proc. of Intl. Conference on Computer-Aided Design*, ICCAD, New York, NY, USA: ACM, 2009, pp.461–467.

[3] J. Pouwelse, K. Langendoen and H.J. Sips. Application-directed voltage scaling. in *IEEE Trans. Very Large Scale Integration Systems (TVLSI)*, vol.11, no.5, pp.812–826, Oct. 2003.

[4] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," in *IEEE TVLSI*, vol.8, no.3, pp.299–316, June. 2000.

[5] K. Choi, W.-C. Cheng, and M. Pedram. Frame-based dynamic voltage and frequency scaling for an MPEG Player. *Journal of Low Power Electronics*, vol. 1, no. 1, pp.27–43, Apr. 2005.

[6] W. Yuan and K. Nahrstedt. Practical voltage scaling for mobile multimedia devices. in *Proc. of the 12th Annual ACM International Conference on Multimedia*, ACM, pp.924–931, NY, USA, 2004.

[7] Y. Gu and S. Chakraborty. Control theory-based DVS for interactive 3D games. in *Proc. of the 45th Annual Conference on Design Automation (DAC)*, New York, USA: ACM Press, 2008, pp.740–745.

[8] S. Sinha, J. Suh, B. Bakkaloglu and Y. Cao. Workload-aware neuromorphic design of the power controller. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 3, pp.381–390, Sep. 2011.

[9] G. Dhiman and T.S. Rosing. System-level power management using online learning. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 28, no. 5, pp.676–689, May. 2009.

[10] M. Pedram. Power optimization and management in embedded systems. in *Proc. of the Asia and South Pacific Design Automation Conference, ASP-DAC'01*, ACM, pp.239–244, Yokohama, Japan, 2001.

[11] V. Pallipadi and A. Starikovskiy. The Ondemand governor. in *Proc. of the Linux Symposium*, 2006.

[12] R. Jejurikar and R. Gupta. Dynamic voltage scaling for systemwide energy minimization in real-Time embedded systems *Proc. of ISLPED'04*, pp.78–81, August, 2004.

[13] H. Jung and M. Pedram. Continuous frequency adjustment technique based on dynamic workload prediction. in *21st IEEE Intl. Conference on VLSI Design*, VLSID, IEEE, 2008, pp.249–254.

[14] K. Choi, R. Soma and M. Pedram. Dynamic voltage and frequency scaling based on workload decomposition. in *Proc. of ISLPED'04*, 2004, pp.174–179.

[15] S. Yue, D. Zhu, Y. Wang, and M. Pedram. Reinforcement learning based dynamic power management with a hybrid power supply. in *IEEE 30th Intl. Conference on Computer Design (ICCD)*, 2012, pp.81–86.

[16] A.K. 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 *Proc. of DAC'14*, pp.1–6, June, 2014.

[17] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu. Achieving autonomous power management using reinforcement learning. *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 2, pp.24:1–24:32, Apr. 2013.

[18] R. Ye, Q. Xu. Learning-based power management for multi-core processors via idle period manipulation. In *IEEE TCAD*, vol.33, no.7, pp.1043–1055, 2014.

[19] BeagleBoard. BeagleBoard-xM Rev C System Reference Manual, 2010. [Online]. Available: http://beagleboard.org

[20] FFmpeg A complete, cross-platform solution to record, convert and stream audio and video. [Online]. Available: https://www.ffmpeg.org/

[21] M.R. Guthaus, M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. MiBench: A free, commercially representative embedded benchmark suite. in *IEEE International Workshop on Workload Characterization (WWC)*, IEEE, pp.3–14, Dec., 2001. [Online]. Available: http://www.eecs.umich.edu/mibench/

[22] FFmpeg Open-source Mozilla Firefox browser. [Online]. Available: https://www.mozilla.org/

[23] BBench Browser benchmarking tool. [Online]. Available: http://bbench.eecs.umich.edu/

[24] T. Jiang, D. Grace and P.D. Mitchell Efficient exploration in reinforcement learning-based cognitive radio spectrum sharing *IET Communications*, vol. 5, Iss. 10, pp. 1309–1317, 2010.

[25] O.S. Unsal, and I. Koren. System-level power-aware design techniques in real-time systems in *Proc. of the IEEE*, vol.91, no.7, pp.1055–1069, July, 2003.

[26] Xilinx Inc. Zynq-7000 All Programmable SoC: ZC702 Evaluation Kit and Video and Imaging Kit (ISE Design Suite 14.2) 2012.

[27] W. Yuan, K. Nahrstedt, S. Adve, D.L. Jones and R.H. Kravets. Design and evaluation of a cross-layer adaptation framework for mobile multimedia systems. in *Proc. SPIE 5019, Multimedia Computing and Networking*, pp.1–13, Jan, 2003.

[28] Hardkernel. Odroid-XU by Hardkernel. [Online]. Available: http://www.hardkernel.com. Last Accessed 10 Dec. 2014.

[29] Y. Ge and Q. Qiu. Dynamic Thermal Management for Multimedia Applications Using Machine Learning. in *Proc. of the 48th Design Automation Conference (DAC)*, New York, USA, pp.95–100, 2011.

[30] S. Sangyoung, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram and N. Chang. Accurate Modeling of the Delay and Energy Overhead of Dynamic Voltage and Frequency Scaling in Modern Microprocessors. in *IEEE TCAD*, vol.32, no.5, pp.695–708, May, 2013.

[31] ARM. Cortex-A8 Technical Reference Manual. [Online]. Available: http://www.arm.com. Last Accessed 7 April 2015.

## Appendix A: Linux Governor Implementation

Fig. 18 shows the block diagram of the Linux power governor implementation consisting of the following interfaces:

### A. Kernel-level Library Interfaces

These interfaces comprise of the scheduler, *CPUfreq* and SYSFS interfaces. The scheduler interface (block A) defines the CPU IDs, system timer and functions to establish governor control of a specific CPU. The *CPUfreq* interface (block B) retrieves information related to processor frequency steps. The SYSFS interface (block C) defines file system variables/constants.

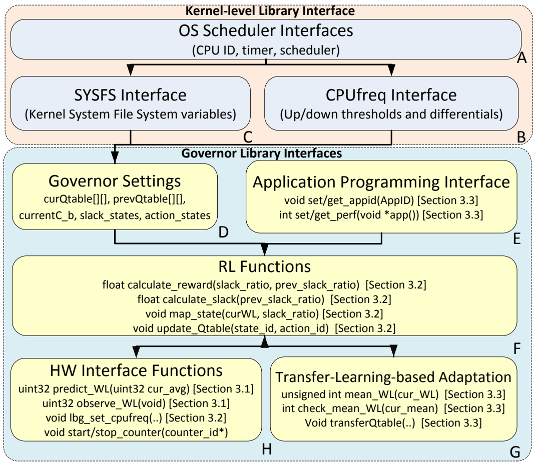

Kernel-level Library Interface
OS Scheduler Interfaces
(CPU ID, timer, scheduler)
A
SYSFS Interface
(Kernel System File System variables)
C
CPUfreq Interface
(Up/down thresholds and differentials)
B
Governor Library Interfaces
Governor Settings
curQtable[][], prevQtable[][],
currentC_b, slack_states, action_states
D
Application Programming Interface
void set/get_appid(AppID) [Section 3.3]
int set/get_perf(void *app()) [Section 3.3]
E
RL Functions
float calculate_reward(slack_ratio, prev_slack_ratio) [Section 3.2]
float calculate_slack(prev_slack_ratio) [Section 3.2]
void map_state(curWL, slack_ratio) [Section 3.2]
void update_Qtable(state_id, action_id) [Section 3.2]
F
HW Interface Functions
uint32 predict_WL(uint32 cur_avg) [Section 3.1]
uint32 observe_WL(void) [Section 3.1]
void lbg_set_cpufreq(..) [Section 3.2]
void start/stop_counter(counter_id*)
H
Transfer-Learning-based Adaptation
unsigned int mean_WL(cur_WL) [Section 3.3]
int check_mean_WL(cur_mean) [Section 3.3]
Void transferQtable(..) [Section 3.3]
G

Fig. 18. Implementation of the proposed approach showing different interfaces

### B. Governor Library Interfaces

These interfaces include the following:

*1) Governor settings (block D):* define the Q-tables (*curQTable* and *prevQTable*), each with its base workload $\mathcal{C}_b$, performance requirement $T_{ref}$ and the state-action pairs.

*2) Application programming interface (block E):* sets up the inter-layer interactions between the application and runtime layers, implemented through a thread-safe handshake signal (called *rts→handshake*), and a SYSFS variable for storing the $T_{ref}$. The kernel initially starts a notification process with the *rts→handshake* signal, which is acknowledged by *set_perf(..)* function to set the $T_{ref}$.

*3) Hardware interface (block H):* defines the interaction and control between the runtime and hardware layers. It processes the necessary prediction and feedback from the CPU performance counters through *predict_WL(..)* and *observe_WL(..)* functions. During observation of the workload, *start_counter(..)/ stop_counter(..)* IO functions are used to start/stop the CPU performance counters using the *ioctl* interface in Linux. The frequency is set for the system using *lbg_set_cpufreq(..)* function.

*4) RL functions (block F):* set up the learning functions in the Q-table (Section III-A2). Based on the predicted workload, *map_state(..)* maps the system's current state, *calculate_reward(..)* calculates reward of a selected VFS action. The calculated reward is then updated through *update_Qtable(..)* function using the Q-values given by (5).

*5) Learning transfer-based adaptation (block G):* is implemented through a set of functions. For detecting workload variations, the short-term mean workload is calculated using *mean_WL(..)* and for performance variations are communicated through API. When a variation is detected, the Q-table is updated and learning is transferred with a new base workload value ($\mathcal{C}_b$) using *transferQtable(..)*.

Since floating point calculations are limited in kernel-level, appropriate scaling was applied to different variables and constants (in Section III). The governor is applied through the Linux command-line as

```
cpufreq_set --cpu 0 --governor ltbg
```

The *cpufreq_set* is a command to administer the governor settings; the *–cpu* option is followed by the CPU ID (.e.g '0') and the *–governor* option specifies the governor name, (e.g. 'ltbg': learning transfer-based governor).

## APPENDIX B

LEMMA I*: For exploration of all $|\mathcal{S}|$ states in a Q-table, each with $|\mathcal{A}|$ actions, the minimum number of explorations required by an EPD-based exploration is given by: $\left(\frac{3}{5}|\mathcal{S}||\mathcal{A}|\right)$, which is less than that required by an UPD-based exploration: $(|\mathcal{S}||\mathcal{A}|)$.*

*Proof:* The exploration of the states with near-zero (i.e. $\pm5\%$) slack values constitute $\frac{1}{5}$th of $|\mathcal{S}|$ and require exploration of all $|\mathcal{A}|$ actions in the Q-table due to (4). The total number of explorations required by these states amounts to $\left(\frac{1}{5}|\mathcal{S}||\mathcal{A}|\right)$. The remaining states constitute $\frac{4}{5}$-th of $|\mathcal{S}|$ and minimally require explorations of only half of the $|\mathcal{A}|$ actions in the Q-table due to relationship between their slacks and actions given by (4). Hence, the total number of explorations required by these states sums up to $\left(\frac{4}{5}\frac{1}{2}|\mathcal{S}||\mathcal{A}|\right)$=$\left(\frac{2}{5}|\mathcal{S}||\mathcal{A}|\right)$. The total number of minimum explorations required for $|\mathcal{S}|$ states by an EPD-based exploration is: $\left(\frac{3}{5}|\mathcal{S}||\mathcal{A}|\right)$. This proves the first part.

The exploration of $|\mathcal{S}|$ states using UPD does not exploit the state-action relationships described in (4). As a result, all slack state-action pairs need to be explored, requiring a total of $(|\mathcal{S}||\mathcal{A}|)$ explorations. This is 40% less than EPD-based explorations. This proves the second part.

PROPOSITION I*: If x% of the higher slack states (i.e. $15\% > |\mathcal{L}_t| > 5\%$) retain their relationships with the chosen actions, the learning transfer algorithm (Algorithm 1) will require minimum $\left[\left(\frac{1}{5}|\mathcal{S}||\mathcal{A}|\right) + \left(\frac{2}{5}(1-x\%)|\mathcal{S}||\mathcal{A}|\right)\right]$ further explorations.*

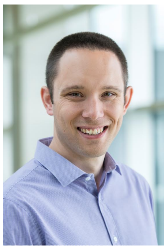
*Proof:* From Lemma I, the minimum number of explorations required for the lower slack states (i.e. $\pm5\%$ slack values) is given by $\left(\frac{1}{5}|\mathcal{S}||\mathcal{A}|\right)$. When x% of the higher slack states retain their relationships with the scaled actions, the minimum number of explorations required by the LT algorithm is given by the fraction of remaining higher slack states by half of the total number of possible actions (due to intrinsic state-action relationships), i.e. $\left(\frac{2}{5}\left(1-x\%\right)|\mathcal{S}||\mathcal{A}|\right)$ (Lemma I). For all slack states, the LT algorithm (Algorithm 1) requires minimum $\left[\left(\frac{1}{5}|\mathcal{S}||\mathcal{A}|\right) + \left(\frac{2}{5}(1-x\%)|\mathcal{S}||\mathcal{A}|\right)\right]$ further explorations to expedite learning. This proves the proposition.

**Rishad A. Shafik** (M'09-) is a lecturer in Electronic Systems at Newcastle University, UK. Dr. Rishad received his Ph.D., and M.Sc. (with distinction) from the University of Southampton in 2010, and 2005; and B.Sc. in Electronic Engineering (with distinction) from the IUT, Bangladesh in 2001. He is one of the editors of the book "Energy-efficient Fault-tolerant Systems," published by Springer USA, and author/co-author of 70+ IEEE/ACM journal and conference articles. His research interests include energy-efficiency and reliability aspects of embedded systems.

**Sheng Yang** received his B.Eng. in Electronic Engineering from the University of Southampton, UK, in 2008, and Ph.D. degree in Electronic Engineering from the same in 2013. In 2007 he worked as an NXP intern for modelling a data hub using SystemC. In 2011 he held an internship with ARM investigating data integrity of flip-flops within microprocessors. Currently, he is working as a research fellow at the University of Southampton. His research interests include low power and fault tolerant design techniques for embedded systems.

**Anup Das** received B.Eng. degree in Electronics and Telecommunication Engineering from Jadavpur University, India, in 2004, and Ph.D. degree in computer engineering from the National University of Singapore, in 2014. He is currently a post-doctoral research fellow at the University of Southampton. From 2004 to 2007, he was with STMicroelectronics Ltd. as an IC design engineer and from 2007 to 2011, he was with LSI Corporation as design-for-test engineer of storage SoCs. His research interests include and reliable design and runtime management of multiprocessor systems.

**Luis A. Maeda-Nunez** is a research student from Mexico. He studied his Bachelor in Electronics Engineering back at ITESM, Mexico. He did his MSc in Microelectronics Systems Design at the University of Southampton where graduated with Distinction in 2011. He started and is currently studying for his PhD on Power Management for Multi-core Processors. His research interests include multi-core architectures, power and thermal management, and Machine Learning.



**Geoff Merrett** (GSM'06-M'09) received the BEng degree (Hons) in Electronic Engineering and the PhD degree from the University of Southampton, UK, in 2004 and 2009. He is currently an Associate Professor in electronic systems at the University of Southampton. His research interests include low-power and energy harvesting aspects of embedded systems, and has published over 90 articles in journals/conferences in these areas. Dr Merrett is a Fellow of the HEA and he was the General Chair of the Energy Neutral Sensing Systems (ENSsys) workshop from 2013 to 2015.

**Bashir M. Al-Hashimi** (M'99-SM'01-F'09) is a Professor of Computer Engineering and Dean of the Faculty of Physical Sciences and Engineering at University of Southampton, UK. He is an ARM Professor of Computer Engineering and Co-Director of the ARM- ECS research centre. His research interests include methods, algorithms and design automation tools for low-power design and test of embedded computing systems. He has published over 300 technical papers, authored or co-authored 5 books and has graduated 31 PhD students.