# Hardware-Software Interaction for Run-time Power Optimization: A Case Study of Embedded Linux on Multicore Smartphones

†Anup Das, †Matthew J. Walker, †‡Andreas Hansson, †Bashir M. Al-Hashimi and †Geoff V. Merrett
†ARM-ECS Research Center, University of Southampton, United Kingdom
‡Research, ARM Ltd, Cambridge, United Kingdom
Email: †{a.k.das,mw9g09,gvm,bmah}@ecs.soton.ac.uk and ‡andreas.hansson@arm.com

***Abstract*—Applications running on smartphones interact with the hardware and the system software differently, resulting in widely varying power consumption and hence thermal profiles. Typically, these smartphone platforms expose some hardware power control features to users, controlled through software governors such as `cpufreq` for dynamic voltage-frequency scaling (DVFS) and `cpuquiet` for dynamic core selection (DCS). Operating systems on these platforms manage these governors conservatively, independent of application's performance requirement. To address this, we propose an alternative approach, which uses reinforcement learning to explore the trade-off between power saving opportunities using DVFS and DCS and application's performance at run-time. The objective is to reduce power consumption, taking into consideration dynamic power, leakage power, and the inter-dependency between temperature and power. The reinforcement learning-based control is validated as a case-study on ARM A15-based nvidia's tegra smartphone through its implementation as a run-time manager (RTM). This RTM interfaces with different hardware performance counters and the embedded Linux Operating System through (1) the `cpuquiet` API to select cores at run-time; and (2) the `cpufreq` API to scale the frequency of active cores. Experiments with mobile and high performance applications demonstrate that the proposed approach achieves an average 22% (7-40%) power reduction compared to existing techniques.**

***Keywords*—Power reduction, temperature minimization, reinforcement learning, cpufreq, cpuquiet**

## I. Introduction

Modern embedded systems feature multiple general purpose cores, which improve application performance by executing its independent threads simultaneously. As more processing cores are integrated in a system, the chip power consumption increases, reducing the battery life [1]. This increase in power consumption also increases chip temperature, triggering reliability concerns [2]. Recent studies show that the leakage power constitutes more than 40% of the total power consumption, being superlinearly dependent on the chip temperature [3]. This has attracted significant attention in recent years [4]–[10].

Two of the most widely accepted system-level design techniques for power optimization are dynamic voltage and frequency scaling (DVFS) [11] and dynamic power management (DPM) [12]. In DVFS, the voltage and frequency are scaled down dynamically to reduce both the active and leakage power consumption, whereas in DPM, the processing cores are shut down (or put into sleep mode) to reduce leakage power. In the context of this paper, we achieve DPM by dynamically controlling the number of active cores and as such, the approach is commonly termed as Dynamic Core Selection (DCS). Operating systems (OSs) such as embedded Linux (eLinux) provide user interfaces for managing both DVFS and DCS. Examples of these interfaces are `cpufreq` [13] for DVFS and `cpuhotplug` [14] for DCS. Typically, `cpuhotplug` is 1000 times slower than `cpufreq`, limiting its use at run-time. Existing studies on run-time management have therefore considered DVFS alone to perform dynamic power optimization[1] [4]–[7]. The commercial version of `hotplug` for embedded systems, called `cpuquiet` [15], provides a low overhead user interface for addition and deletion of cores at run-time. The `cpuquiet` and the `cpufreq` APIs are widely used for run-time power management in OSs. Examples include the ARM Intelligent Power Allocation (IPA) and ARM Energy Aware Scheduler (EAS). Our approach complements these techniques by exploring the trade-off between performance loss and power saving opportunities using machine learning.

Recently, performance impact of DVFS and DCS have been studied using high level application graph models (directed acyclic graphs or synchronous data flow graphs) representing static workload scenarios [9], [10]. The power-temperature inter-dependency is either not incorporated or the influence of ambient temperature is not factored. From a practical aspect, applications running on embedded systems interact with the OS and the hardware differently, resulting in widely varying thermal and power profiles. The performance requirement also differs from one application to another, requiring application-specific voltage-frequency settings. Additionally, the nature of cross-layer interaction and the performance requirement varies within application execution, as observed for instance when switching from 4K resolution video to a high-definition (HD) video. These intra- and inter-application variations present a dynamic scenario to determine the minimum number of cores and their operating point at run-time. To address this, we propose a reinforcement learning-based run-time approach that adapts to intra- and inter-application variations by adding or deleting cores at run-time using the `cpuquiet` governor, and controlling the voltage and frequency of operation using the `cpufreq` governor. The objective is to explore the trade-off between an application's performance (specified as deadline or throughput constraint) and power saving opportunities. Following are our key contributions:

- a reinforcement-learning based approach for power management of embedded systems, considering the inter-dependency of temperature and power;
- integrating DCS and DVFS together in a run-time framework, considering both dynamic and leakage power components simultaneously; and
- adapting to intra- and inter-application variations in order to deploy an application-specific strategy for thermal-aware power management.

Remainder of this paper is organized as follows. The problem formulation is discussed next in Section II along with the motivation for a solution using machine learning. The proposed approach is described in Section III and its evaluation

[1]Some OS- based approaches achieve DPM by increasing the idleness of cores at run-time [4], [8]. These approaches reduce power consumption only if an application's idle period is greater than the minimum idle time [3], which is difficult to determine at run-time.

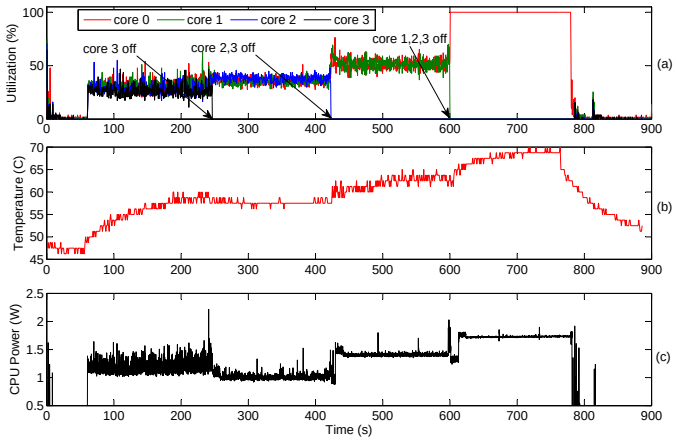


Fig. 1. Utilization, temperature and power variation with changes in the number of active cores.

case-study in Section IV. Finally, the paper is concluded in Section V.

## II. Problem Formulation and Motivation

### A. Processor Power Consumption

The dynamic power of a processor is directly proportional to the frequency ($f$) of operation and quadratically proportional to the voltage ($V$), i.e. $P_d \propto f \cdot V^2$. The static power ($P_s$) is given by [3], i.e. $P_s = V \cdot I_{leak}$, where $I_{leak}$ is the leakage current. As discussed in [3], out of the five leakage components in modern CMOS transistors, the only temperature-dependent dominant leakage component is the sub-threshold leakage current, which is given by

$$I_{sub} = V \cdot I_o \times \left[ AT^2 e^{\frac{\alpha V + \beta}{T} + Be^{\gamma V + \delta}} \right] \quad (1)$$

where $T$ is the temperature, $I_o$ is the leakage current at the reference temperature, and $A, B, \alpha, \beta, \gamma, \delta$ are the technology dependent constants. Clearly, the sub-threshold leakage is super-linearly dependent on the temperature.

### B. Processor Temperature

The temperature of a core is related to its power dissipation according to the following equation [16].

$$C\frac{dT(t)}{dt} + G\left(T(t) - T_{amb}\right) = P(t) = P_d + P_s \quad (2)$$

where $C$ is the thermal capacitance, $G$ is the thermal conductance, $t$ is the time, $T_{amb}$ is the ambient temperature, $T(t)$ is the instantaneous temperature and $P(t)$ is the instantaneous power, which is composed of the dynamic and the leakage components. As seen from Equations 1-2, there is an inter-dependency between temperature and power.

### C. Interplay of DCS and DVFS

To demonstrate the interplay of DCS and DVFS, we conducted an experiment on nvidia's smartphone platform (the Jetson development board) with a multithreaded application. The application is executed for several iterations; each iteration is accompanied by a deadline, which serves as the performance requirement. At each iteration, six threads are spawned with each thread performing basicmaths, crc and fft operations in series but on different data set. A simple proportion-integral (PI) controller is used as a Kernel module for the eLinux

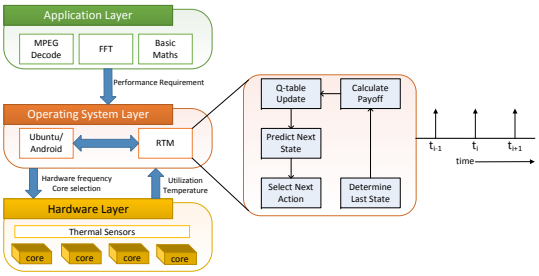


Fig. 2. Three-layered representation of an embedded system with the proposed approach indicated as RTM.

running on the platform to determine the operating point. Specifically, the control algorithm scales down the operating frequency whenever there is slack in the application. In this context it is worth mentioning that eLinux allow scaling the frequency only; the voltage is scaled proportionately.

With this setup, Figure 1 plots the utilization, temperature and the CPU power consumption as the number of cores is decreased from 4 to 1 (left to right of the figure) using the `cpuquiet` API implementing `cpuhotplugging`. The following observations can be made from this figure.

**Observation 1:** Utilization of the active cores increases with decrease in core count. In the interval 50s to 250s in Figure 1, all four cores are active, resulting in an average utilization of 45% across the cores. In the interval 250s - 425s, three cores are active and the average utilization is 47%. In the interval 425s - 600s, core 0 and core 1 are active with an average utilization of 60% for the two cores. Finally, in the interval 600s - 800s, only one core (core 0) is active, resulting in an utilization of 100% for core 0.

**Observation 2:** The temperature and total power consumption increases with decrease in the core count. In our earlier work [17], we have shown that the processor utilization correlates to a reasonable accuracy with the dynamic power consumption for ARM A15 cores. This is evident from the results obtained with 1, 2 and 3 active cores, where the power consumption increases with a reduction of the active cores. It is worth noting that with 1 core, the frequency is also higher (due to the deadline requirement) contributing further to the dynamic power. However, when all 4 cores are active (interval 50s to 250s ), the power consumption is higher than that obtained with 3 active cores. This is due to high active power as compared to that of deep sleep mode when it is hotplugged.

To conclude, the power consumption of an application is dependent on the number of active cores, application's cross-layer interactions, the CPU utilization and the thermal profile. Some of these dependencies are not known prior to executing the application on the hardware. Therefore, no single policy (DCS or DVFS) can guarantee minimum power for all applications. Application workload guides the selection of the cores and their voltage-frequency values. Additionally, due to the large number of unknown dependencies, unsupervised machine learning, in particular reinforcement learning is best suited for the workload-specific power optimization problem.

## III. Run-time Manager for eLinux

The proposed approach is validated through its implementation as run-time manager (RTM) for eLinux. Typically, embedded systems are not equipped with power monitors. To implement a closed-loop power control (i.e. evaluating the impact of an applied action), we used the CPU power

model [17], which estimates the power consumption of a workload by reading hardware performance counters. The leakage power consumption is calculated using the technology dependent parameters of Equation 1. These parameters are characterized for the board, as discussed in Section IV. The temperature for a given workload is measured by reading the on-chip thermal sensor.

Figure 2 shows the three-layered representation of an embedded system. The top most layer is the *application* layer with active applications; the middle layer is the OS layer (eLinux), coordinating application execution on the hardware; the bottom layer is the hardware layer consisting of multicore processors. Interactions among these layers are indicated with arrows. Our approach is implemented as part of eLinux (indicated as *RTM*). The RTM, which uses Q-learning algorithm (a variant of reinforcement learning), repeatedly observes the current state of the system, and selects an action. The selected action changes the system 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 effect its action 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 the payoff, which is stored in a table (termed Q-table). Eventually, the RTM learns to select the best action in order to maximize the long-term sum of future payoffs.

The RTM works at the system time ticks (indicated in the figure). The learning algorithm proactively manages the power consumption, i.e. it takes action to prevent the system from reaching a high power state. Workload prediction is inherent to this algorithm, i.e. at time $t_i$, the algorithm predicts the workload for the next interval to select the best action. At time instant $t_i$, the RTM performs the following steps:

- 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.

**Payoffs:** The payoff at time $t_i$ is computed as

$$R(t_i) = \begin{cases} w_t \times [P_{max} - P_{avg}(t_{i-1} \rightarrow t_i)] & \text{if } L_i \geq L_c \\ w_s \times (L_i - L_c) & \text{otherwise} \end{cases} \quad (3)$$

where $P_{max}$ is the power corresponding to the highest frequency set on all cores, $P_{avg}(t_{i-1} \rightarrow t_i)$ is the average power in the interval $t_{i-1} \rightarrow t_i$, $L_i$ is the performance in this interval, $L_c$ is the performance constraint, and $w_t, w_s$ are the weights. The equation is interpreted as follows: if the performance obtained in an interval is greater than the performance constraint, the power overhead is used to compute the payoff; otherwise, the negative of the performance slack is used as the payoff. It is to be noted that voltage, frequency and temperature are incorporated in the computation of $P_{avg}$.

**System State:** The state of an embedded system is represented using CPU cycle count i.e., the system state $s_i$ at time $t_i$ is given by $s_i = \sum_j CPU\_CYCLES(t_{i-1} \rightarrow t_i)$, where $j$ is the number of active cores. The CPU cycle count is a real number; to limit the state space, each state $s_i$ is discretized to one of the $N_s$ levels and is indicated as $\hat{s}_i$. The discrete states form the rows of the Q-table.

**System Action:** An action for the RTM consists of (1) core selection and (2) frequency of the active cores. In typical mobile systems, all processing cores are on the same voltage domain, allowing chip-wide DVFS. The $k^{\text{th}}$ action is therefore, represented as $a_k = \langle c_1^k \; c_2^k \; \cdots \; c_{N_c}^k \; f_k \rangle$, where $c_j^k$ is a binary indicator to indicate if core $c_j$ is enabled for action $a_k$, $f_k$ is the frequency selected for all active cores, and $N_c$ is the number of cores. The total number of actions is $N_a = 2^{N_c} * N_f$, where $N_f$ is the number of frequencies. These actions form the columns of the Q-table.

**ALGORITHM 1:** Q-learning implemented in the RTM

**Input**: Average temperature $T_i$ in the interval $t_{i-1} \rightarrow t_i$ and CPU cycle count $\sum_j CPU\_CYCLES(t_{i-1} \rightarrow t_i)$ in the interval
**Output**: Core selection and hardware frequency
1 Calculate Payoff (Equation 3);
2 Update Q-table entry (Equation 4);
3 Predict Next State (Equation 6);
4 Select Action (Equation 7);
5 Map action to core selection and hardware frequency;

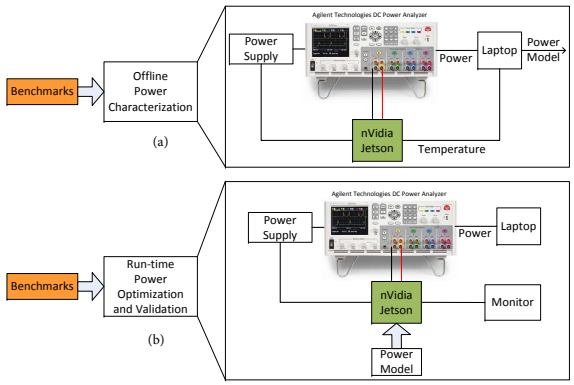


Fig. 3. Setup for power characterization and use at run-time.

`cpuquiet` [15] allows auto hotplugging i.e., dynamically selecting which cores need to be enabled for an application. Following are the sequence of events that are carried out for core $c_j$, when $c_j^k$ changes from 1 to 0 i.e., $c_j^k : 1 \rightarrow 0$.

- The event `CPU_DOWN_PREPARE` is sent to the kernel.
- Kernel migrates running processes on $c_j$ to other cores.
- Kernel invokes architecture specific `_cpu_disable()`.
- The event `CPU_DEAD` is sent to offline $c_j$.

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

$$Q(\hat{s}_{i-1}, \hat{a}_{i-1}) = Q(\hat{s}_{i-1}, \hat{a}_{i-1}) + \alpha \times R(t_i) \quad (4)$$

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

$$\alpha = \begin{cases} 1 & \text{for } 0 \leq N < N_{explore} \\ 2^{(N_{explore} - N)} & \text{for } N_{explore} \leq N < N_{exploit} \\ 0 & \text{for } N \geq N_{exploit} \end{cases} \quad (5)$$

where $N$ is the number of visits, and $N_{explore}$/$N_{exploit}$ are the constants indicating the limits of the Q-learning stages, i.e., exploration, exploration-exploitation and exploitation.

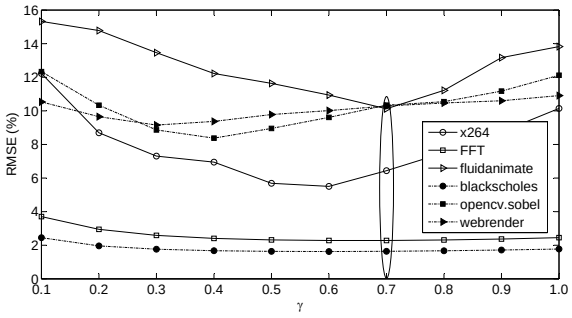


Fig. 4. Root mean square workload prediction error (RMSE) for different $\gamma$.

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

$$p_{i+1} = \gamma \times \hat{s}_i + (1 - \gamma) \times p_i \tag{6}$$

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

$$a_{i+1} = \texttt{argmax}\ \text{Q-table}(\hat{p}_{i+1}, :) \tag{7}$$

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. Algorithm 1 summarizes the Q-learning algorithm.

## IV. Case Study: eLinux on Tegra K1 SoC

We present a case-study of the hardware-software interaction with eLinux on nvidia's Jetson board featuring a Tegra K1 SoC [18] with a quad-core ARM Cortex-A15 CPU. The platform supports 22 different frequencies (50MHz to 2.32GHz) and integrates a CPU thermal sensor for temperature measurement. A set of multithreaded benchmarks from from MiBench [19], PARSEC and the SPLASH2 [20] suites are used to build a workload-dependent CPU power model [17]. The modeling setup is shown in Figure 3(a), where performance counters corresponding to a workload are used together with voltage, frequency and temperature to correlate (using a nonlinear fit) with the power consumption recorded from the DC power analyzer from Agilent Technologies (N6705B). Benchmarks used for building the power model are different to those used for validating the reinforcement learning-based RTM approach.

### A. Evaluation of the Proposed RTM

*1) Power Estimation Error:* Using the setup of Figure 3, the average power estimation error is 3.5%, with a maximum of 6.1% for database manipulation application. Detailed results on power estimation accuracy are presented in [17].

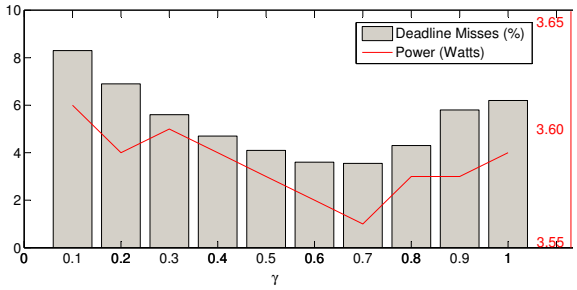


Fig. 5. Effect of workload under-prediction.

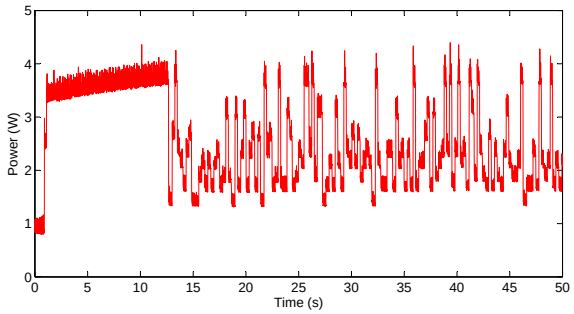


Fig. 6. Exploration phase of the Q-learning.

*2) Workload Prediction Error:* The smoothing factor $\gamma$ defines the relative importance of the predicted workload as compared to the actual workload of the prior frames. Figure 4 plots the root mean square prediction error (RMSE) of the workload (CPU statistics) by varying $\gamma$ (Equation 6) for six applications. For some applications such as FFT and blackscholes, the RMSE is lower and relatively invariant with $\gamma$ as compared to applications such as x264 and fluidanimate. This is because, the workload for FFT and blackscholes are relatively static (lower variations across frames) and therefore, these workloads can be predicted with reasonable accuracy as compared to that of x264 and fluidanimate. It can also be noted that initially, the RMSE decreases with an increase in $\gamma$ implying that the prediction accuracy increases. However, beyond $\gamma$ = 0.7, the prediction error increases. $\gamma$ = 0.7 produces the least prediction error for most applications.

Figure 5 plots the effect of varying the smoothing factor $\gamma$ on the number of deadline misses (expressed as percentage of the total frames) and the power consumption (in watts) for the ffmpeg application used to play a 1080p video. As $\gamma$ increases, the number of workload miss-predictions (over/under) decreases until $\gamma$ = 0.6-0.7, beyond which the miss-prediction again increases. A lower number of workload under-prediction translates to a lower number of frames missing deadline[2]. It is to be noted that in most video decoders, frames missing deadline are usually dropped. This results in glitch in the output video and therefore, degrades quality of user experience. Similarly, a lower number of workload over-prediction translates to lower power consumption. As seen from the figure, a $\gamma$ values of 0.6-0.7 yields the best result in terms of the number of deadline misses and power consumption. A similar trend is observed for all other applications.

*3) Stages of Q-Learning:* The Q-learning algorithm used in our approach has three phases – an initial exploration

[2]Typically, the display subsystem has a buffer of one frame. Thus, the deadline for a frame is equal to 42 ms for a 24 fps video.

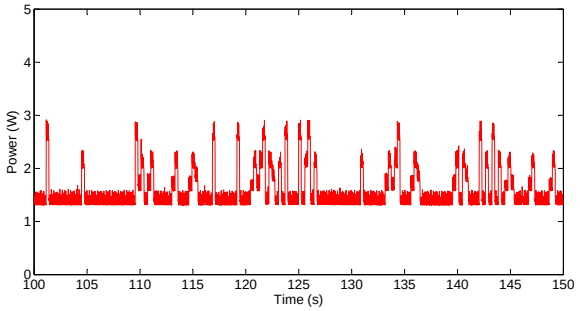


Fig. 7. Exploitation phase of the Q-learning.

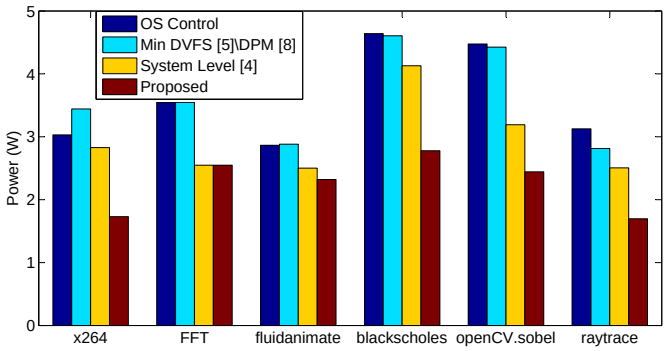


Fig. 8. Power for 6 applications: proposed approach vs [4], [5], [8].

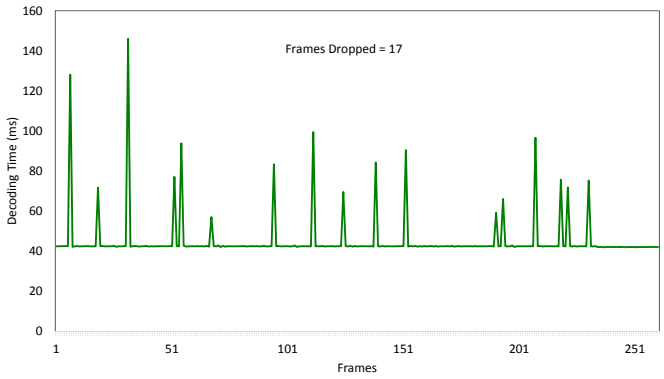


Fig. 9. Frame decoding time using ffmpeg playing a 1080p video.

phase, followed by an exploration-exploitation phase and finally, the exploitation phase. Figures 6 and 7 plot the power obtained using the proposed RTM during the exploration and the exploitation phase. In the exploration phase (Figure 6), the algorithm explores different actions (`cpuquiet` and `cpufreq`) to determine the most appropriate control for the application workload. The average power in this stage is 2.8W. The power consumption using the operating system's default `cpuquiet` governor is also similar (2.75W). However, as the algorithm enters the exploitation phase (Figure 7), best actions are exploited for a given workload. The average power consumption in this stage is 1.6W (1.15W savings compared to the default `cpuquiet` governor). This improvement clearly demonstrates the advantage of the proposed approach over the operating system controlled DCS-DVFS technique. Further evaluation with other state-of-the-art approaches is provided in the following section.

### *B. Power Improvement using the RTM*

Figure 8 reports the power improvement of the proposed approach in comparison to state-of-the-art approaches. Specifically, we compare our approach with the OS-controlled approach (a combination of `cpuquiet` and `cpufreq`), the minimum of the power results obtained using the DVFS only technique of [5] and the DCS only technique of [8], and the system level technique of [4] that selects between DCS and DVFS policies based on application. As seen from the figure, the min DVFS/DCS approach performs significantly better than the OS controlled approach for some applications, such as the raytrace, while the OS-controlled approach is better for the x264 application. In comparison to both these approaches, the technique of [4] minimizes the power consumption by an average 16%. This result is consistent to that reported in [4]. The proposed approach achieves a similar power consumption as [4] for the FFT application, which has a static workload. However, for all other applications, the result using the proposed approach is significantly better, achieving on average 23% further power improvement compared to [4].

### *C. Performance Trade-off using the RTM*

Figure 9 plots the decoding time taken by the ffmpeg application playing a 1080p video at 24 fps resolution. Results are reported for the first 260 frames of this video (approximately 11 sec). As can be seen, the decoding time occasionally exceeds 50 ms causing these frames to be dropped by ffmpeg application. As seen from the figure, the ffmpeg application drops 17 out of 260 frames. On average, the decoding time for the displayed frames is 42.3 ms (instead of 41.67 ms requirement of the video). However, this increase in decoding time is due to processor slowdown for power savings without perceivable degradation of video quality. This highlights the fact that the proposed approach reduces power consumption by trading-off 1.52% performance.

To summarize the result for other applications, we conducted experiments with twenty different applications from the benchmark suites discussed before. Figure 10 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 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 power savings (average 23%) 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 power savings opportunities. The highest performance slack that remains to be exploited is 3% (in the figure, the number of application with performance variation of 4% or above is zero).

### *D. Thermal Improvement using the RTM*

As can be seen from Equation 2, the temperature of processing cores is dependent on the power consumption, which in turn depends on the temperature. To address this inter-dependency of temperature and power, both these metrics are incorporated in computing the payoff (specifically, as $P_{avg}$ of Equation 3). To signify the thermal improvement achieved

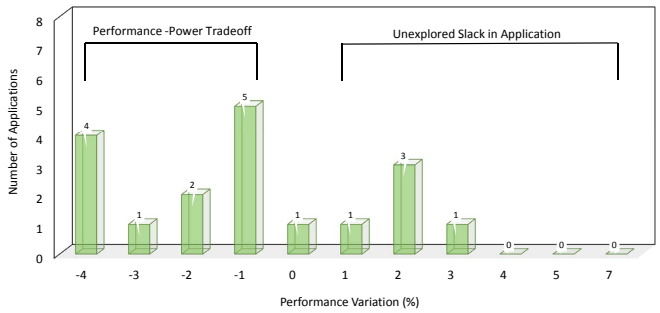


Fig. 10. Performance summary across 20 different applications.

TABLE I. THERMAL IMPROVEMENT FOR FFT APPLICATION.

| Techniques | Average Temperature | Peak Temperature |
|---|---|---|
| OS Controlled | 76.6°C | 82 |
| System level [4] | 69.9°C | 79°C |
| Proposed | 62.1°C | 70°C |

using the proposed approach, Table I reports the average and peak temperature in comparison to some state-of-the-art approaches. The FFT application is used for demonstration. As can be seen, the proposed thermal-aware power-optimization approach reduces average temperature by 15°C and the peak temperature by 12°C as compared to the OS controlled approach. In comparison to the system level technique of [4], the improvements are 8°C and 9°C, respectively. A similar improvement is observed for all other application.

### *E. RTM Power and Timing Overhead*

Figure 11 plots the average number of invocations of the `cpuquiet` and the `cpufreq` APIs during execution of five applications. As can be seen, for the X264 decoder, the proposed approach invokes the `cpuquiet` API four times during execution for DCS, with the `cpufreq` API being invoked an average 12 times for DVFS during each invocation of the `cpuquiet` API. Similarly, results for other applications can be interpreted. It is interesting to note that for the FFT application, the workload is static and therefore the proposed approach performs DCS only once. On the other end for x264 application, the proposed approach performs DCS four times due to the dynamic nature of its workload. It can also be noted that although 22 frequency levels are supported on the platform, the proposed approach explores a subset of these levels due to the specified performance requirement. For application such as fluidanimate, the number of explored DVFS levels is much higher due to its relaxed deadline than that for FFT and x264 applications. Finally, the proposed RTM constitutes between 0.05% to 0.4% of the frame processing time for all applications. In terms of power overhead, frequency switching results in an overhead of 0.1W to 0.6W and CPU hotplugging has an overhead of an average 0.7W. These are the instantaneous powers recorded directly from the power analyzer.

## V. CONCLUSIONS

We proposed reinforcement learning-based hardware-software interaction for run-time power optimization. Power reduction is achieved by reducing the number of active cores and down-scaling frequency of theses active cores, trading-off performance (in terms of dropped frames), while still maintaining a satisfactory quality-of-service. A case study is provided on nvidia's smartphone to demonstrate power savings using such interactions.

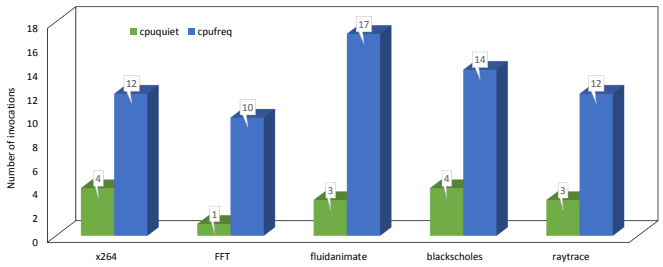


Fig. 11. Number of invocations of `cpuquiet` and `cpufreq` for five applications.

## 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). The data for this paper can be found at 10.5258/SOTON/377395.

## REFERENCES

[1] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Power challenges may end the multicore era," *Communication of the ACM*, vol. 56, no. 2, pp. 93–102, 2013.

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

[3] Y. Liu, R. P. Dick, L. Shang, and H. Yang, "Accurate temperature-dependent integrated circuit leakage power estimation is easy," in *Conference on Design, Automation and Test in Europe*, 2007.

[4] 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, 2009.

[5] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems*, vol. 18, no. 2, 2013.

[6] Y. Wang, Q. Xie, A. Ammari, and M. Pedram, "Deriving a near-optimal power management policy using model-free reinforcement learning and bayesian classification," in *Design Automation Conference*, 2011.

[7] D.-C. Juan and D. Marculescu, "Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors," in *International Symposium on Low Power Electronics and Design*, 2012.

[8] 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, 2014.

[9] V. Devadas and H. Aydin, "On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications," *IEEE Transactions on Computers*, vol. 61, no. 1, 2012.

[10] M. E. T. Gerards and J. Kuper, "Optimal dpm and dvfs for frame-based real-time systems," *ACM Transactions on Architecture and Code Optimization*, vol. 9, no. 4, 2013.

[11] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli, "Dynamic voltage scaling and power management for portable systems," in *Design Automation Conference*, 2001.

[12] L. Benini, A. Bogliolo, and G. De Micheli, "Dynamic power management of electronic systems," in *International Conference on Computer-Aided Design*, 1998.

[13] J. Hopper *et al.*, "Using the linux cpufreq subsystem for energy management," *IBM blueprints*, 2009.

[14] Z. Mwaikambo, A. Raj, R. Russell, J. Schopp, and S. Vaddagiri, "Linux kernel hotplug cpu support," in *Linux Symposium*, vol. 2, 2004.

[15] P. De Schrijver *et al.*, "cpuquiet: Dynamic cpu core management," *Linux Plumbers Conference*, 2012.

[16] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Transactions on Architecture and Code Optimization*, vol. 1, no. 1, 2004.

[17] M. Walker, A. Das, G. 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.

[18] N. Corpration, "Nvidia tegra mobile processor," *URL http://www. nvidia. com/object/tegra. html*, 2013.

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

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