Machine Learning for Run-Time Energy Optimisation in Many-Core Systems

Dwaipayan Biswas[†], Vibishna Balagopal[†], Rishad Shafik[‡], Bashir M. Al-Hashimi[†], and Geoff V. Merrett[†]
[†] Department of Electronics and Computer Science, University of Southampton, United Kingdom, SO17 1BJ
[‡] School of Electrical and Electronic Engineering, Newcastle University, United Kingdom, NE1 7RU
Email: [†]{db9g10, vb1a15, gvm, bmah}@ecs.soton.ac.uk, [‡] rishad.shafik@ncl.ac.uk

***Abstract*—In recent years, the focus of computing has moved away from performance-centric serial computation to energy-efficient parallel computation. This necessitates run-time optimisation techniques to address the dynamic resource requirements of different applications on many-core architectures. In this paper, we report on intelligent run-time algorithms which have been experimentally validated for managing energy and application performance in many-core embedded system. The algorithms are underpinned by a cross-layer system approach where the hardware, system software and application layers work together to optimise the energy-performance trade-off. Algorithm development is motivated by the biological process of how a human brain (acting as an agent) interacts with the external environment (system) changing their respective states over time. This leads to a pay-off for the action taken, and the agent eventually learns to take the optimal/best decisions in future. In particular, our online approach uses a model-free reinforcement learning algorithm that suitably selects the appropriate voltage-frequency scaling based on workload prediction to meet the applications' performance requirements and achieve energy savings of up to 16% in comparison to state-of-the-art-techniques, when tested on four ARM A15 cores of an ODROID-XU3 platform.**

Keywords—Energy Management; Dynamic Voltage/ Frequency Scaling; Reinforcement Learning; Multi-Core Systems;

I. Introduction

Energy efficiency and high performance continue to be prime research objectives for processor designers of multi-core/many-core platforms [1]. One approach to energy minimization is through Dynamic Voltage Frequency Scaling (DVFS), enabling on-the-fly optimisation of frequency (*F*) and voltage (*V*). This can yield a cubic reduction in dynamic power consumption while maintaining a required Quality of Service (QoS) [2]. DVFS is controlled by the system software, examples of which include Linux's power governors. DVFS techniques can be broadly classified into two types: offline and online. The majority of the research has focused on offline mechanisms, which pre-characterize the applications; the profiled workloads/tasks (pertaining to the application) are used during run-time to control the Voltage-Frequency (*V-F*) levels to achieve energy minimization [3], [4]. However, a lack of run-time adaptation to variations in workload characteristics and changes in application performance requirements renders these techniques less effective. Online techniques control the *V-F* levels based on processor workloads and can be either reactive [5] (where *V-F* is controlled depending on historical CPU workloads) or proactive [6], [7] (the predicted workload is used to control the *V-F* levels for minimising energy). In reactive approaches, if the CPU workload is lower/higher than a pre-defined threshold, subsequently, a decreased/increased *V-F* is used. However, in proactive approaches, the impact of control depending on a predicted workload scenario is observed and adjusted through feedback from hardware performance monitors.

Online approaches using machine learning algorithms learn the *V-F* settings required for an application to minimize energy consumption while application generated processor workloads vary. However, the majority of online approaches [8], [9] do not consider changing application performance requirements. Processor workloads are exercised differently depending on the application tasks being executed. Applications running on modern embedded systems incur workload and performance variations which change dynamically depending on the computation, challenging existing energy management techniques. Moreover, existing online approaches [10] use a single run-time formulation of *V-F* scaling for a given performance requirement and hence fail to adapt to the dynamic variations during application execution.

In this paper, we report on a biologically inspired run-time framework for management power in many-core systems. The framework is based upon the Reinforcement Learning (RL) approach described in [11], [12]. The framework invokes workload prediction and appropriate *V-F* control to achieve energy minimisation for applications executed on a multi-core hardware platform. The challenges of a prediction-control problem are met using RL which is inspired from the biological process of the human brain's reaction to changes in the surrounding environment [13]. We use Q-learning (a variant of RL [13]) for developing the RTM framework, where the RTM residing in the operating system controls cross-layer interactions between the application and hardware layers by taking decisions which lead to a pay-off over time. We demonstrate our proposed approach by executing various applications: MPEG4 decoding, an FFT, and PARSEC and SPLASH2 benchmarks. These execute on the four ARM A15 cores of an ODROID XU3 platform, showing improved energy consumption while adapting to performance and workload variation.

This work was supported in part by the Engineering and Physical Research Council (EPSRC) for their financial support under grant numbers EP/K034448/1 "PRiME: Power-efficient, Reliable, Many-core Embedded systems" (www.prime-project.org) and EP/L000563/1 "Continuous on-line adaptation in many-core systems: From graceful degradation to graceful amelioration". Experimental data used in this paper can be found at DOI: 10.5258/SOTON/404064 (http://doi.org/10.5258/SOTON/404064).

II. Efficient Run-Time Energy Management

A cross-layer view of an embedded system, highlighting the interaction between the application, run-time and hardware layers, is shown in Fig. 1. The *Application* layer is the top layer where applications are being executed. Each application comprises of a series of tasks/workloads being executed at given time intervals (decision epochs) where the specific performance requirements of each task are specified through an Application Programming Interface (API). The *Run-time* layer in the middle represents the operating system (OS) and system software, responsible for coordinating an application's execution on the hardware platform. The bottom *Hardware* layer comprises processing elements (e.g. four ARM A15 processor cores) and their associated peripherals [14]. The run-time management algorithm (RTM) is implemented as a power governor in the OS, where its primary responsibility is energy minimisation through optimised control of hardware levers (*V-F* settings) at regular decision epochs for given performance requirements.

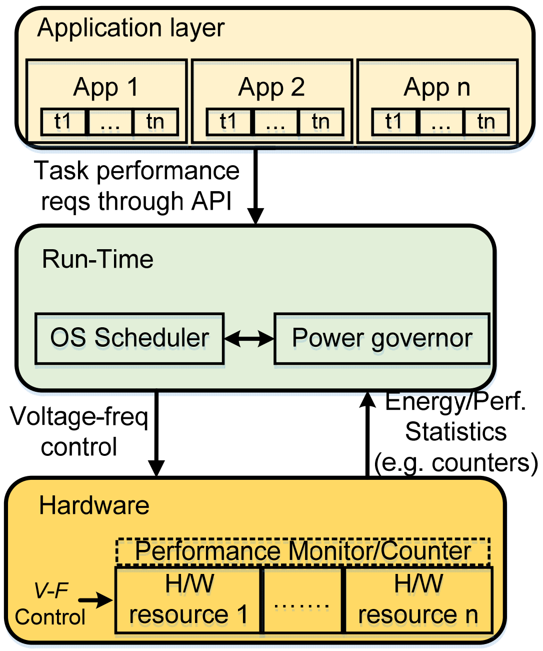

Fig. 1. Cross-layer interaction between the *Application* (comprising of tasks/workloads), *Run-time* and *Hardware* layers.

The key interactions between the *Run-time* (within the OS layer) and *Hardware* layers could be represented conceptually as closed loop system, illustrated in Fig. 2(a). Here, the data extracted from the hardware (performance monitoring units (PMUs), power measurements, etc.) and the performance requirements of the application (frames per second, deadlines, latencies, etc.) act as inputs to the RTM which operates based on a learning algorithm and can make intelligent decisions leading to energy savings. This closed loop system is motivated by the operation of the human brain (in this case acting as an agent) interacting with the surrounding environment, whereby the brain continually learns from observations and takes appropriate actions which yield a reward (positive or negative depending on the action); see Fig. 2(b). Over time, the goal is to reinforce good decisions to maximize positive rewards [13].

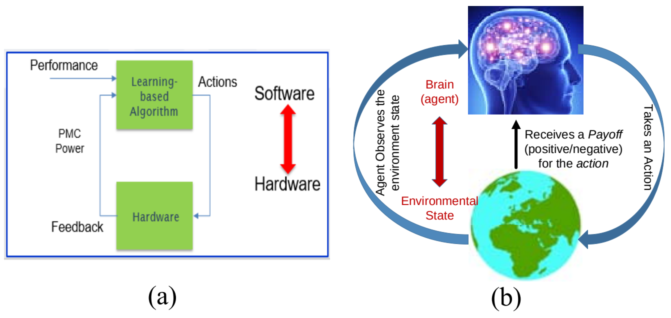

Fig. 2. (a) Closed-loop interaction of RTM with Hardware (b) Closed-loop interaction of Human Brain with Environment [15]

The RTM uses the Q-learning algorithm, which works by observing the current state of a system and selecting an action, leading to a change in the system state. As a result, an immediate numerical pay-off is computed which qualifies the action based on a positive or negative return. Positive pay-offs are termed as profits and negative pay-offs are termed as penalties. The RTM's objective is to learn and select actions to maximize the long-term sum or average of future pay-offs. Initially, the RTM is agnostic about its actions on the system states and the related pay-offs, storing its decisions in a look-up table (referred to as a *Q*-table). This is termed the *exploration phase*, which is followed by the *exploitation phase* where the RTM evaluates those decisions which led to a positive pay-off, and finally, with complete knowledge of the inherent characteristics of the tasks being executed, always selects the best actions (which yielded the highest pay-off) for a given system state. The mapping of the different components of the Q-learning algorithm to our run-time framework is discussed in detail in the following sub-sections.

The RTM works at system time ticks referred as t_{i-1}, t_i and t_{i+1}, where the respective intervals are also termed as decision epochs. The role of the RTM at time t_i can be enlisted as:

(1) computes pay-off for the interval t_{i-1} - t_i;

(2) updates the *Q*-table corresponding to the state-action for t_{i-1};

(3) selects an action for the interval t_i - t_{i+1} based on a predicted value of state.

The RTM proactively manages energy and hence the next state is predicted and an appropriate action is chosen prior to the system reaching the state. Each of these main steps are explained in further detail in the following subsections.

A. State Prediction and Q-table

Predicting the state of the system is a key step in RL and in our methodology the expected workload is classified into a system state at the beginning of each decision epoch [8], [9]. The state of the system is represented using the CPU Cycle Count (*CC*), obtained using the performance monitoring unit. The choice of *CC* as a preferred workload parameter over other parameters such as memory accesses, cache misses, or instruction rate is motivated by the fact that it directly presents a measure of CPU activity while executing instructions of a specific task. Workload prediction schemes generally use adaptive filters [16], which fall short of fine-grained prediction due to a lag inherent in the filtering technique. Hence, they have limited utility for applications incurring dynamic workload changes [3], [17]. In our approach, the current CPU workload is estimated based on a history of past workloads, and this is mapped to a system state based on the current performance. The states form the rows of the look-up table, referred to as the *Q-table* and is determined using the *CC* and current performance (average slack, *L*). The size of the *Q-table* is limited by discretising the range of workloads (slack and cycle count) into *N* levels. Here we have used *N* as 5 in view of a pre-characterisation of the applications, performed to ascertain the inherent workload variability (*viz. design space exploration*). The algorithm used for state prediction is the Exponential Weighted Moving Average (EWMA) [18], whereby the predicted workload for the t_{i+1}-th decision epoch, CC_{i+1} is given by:

$$CC_{i+1} = \gamma \times actualCC_i + (1-\gamma) \times predCC_i \quad (1)$$

where, γ is the smoothing factor. The predicted state for the interval (t_i - t_{i+1}) is determined from the previous state [predicted for the interval (t_{i-1} - t_i): $predCC_i$] and the actual state during that interval ($actualCC_i$). However, workload prediction through (1) still provides mispredictions at run-time due to variations in workload. The impact of such mispredictions on the corresponding *V-F* controls is discussed in Section III-B. Therefore, for each predicted workload (CC_{i+1}) and the current performance (L_i), the system state is mapped to one of the discrete *N* levels.

The size of the *Q*-table in terms of the total number of *state–action* pairs ($|A\{V, F\}| \times |S\{CC,L\}|$; where $|S|$ represents all states in *Q*-table, each having $|A|$ actions) is important for the RL algorithm, and is carefully chosen as it influences the trade-off between learning overhead and the energy minimization achieved. With the given state prediction and *Q*-table formation, the RL algorithm carries out exploration and evaluation of the *V-F* controls as discussed below.

B. Exploration

This is a crucial phase which involves learning of appropriate actions (*V-F* controls) depending on system states. The intuitive relationship between the *state-action* pairs is defined using a discrete Exponential Probability Distribution (EPD) function for the selection of an action which is in contrast to the commonly used random selection policy based on a Uniform Probability Distribution (UPD) [19]. The EPD is expressed as:

$$p(a_i)_{a_i \in A\{V,F\}} = \lambda \exp\left[-\lambda F_a \beta L\right] \quad (2)$$

where λ represents the uniform probability of actions, *F* is the operating frequency for action *a*, β is a constant, *L* is the slack measurement and *A* represents the set of all possible actions. For values of *L* close to zero, the Exponential Probabilities (EP) guided by λ are almost uniform. The Q-value at the start of a decision epoch t_{i+1}, corresponding to a selected *V-F* action is updated with respect to Bellman's optimality equation:

$$Q(s_{i+1}, a_i) = Q(s_i, a_i)(1-\alpha) + \alpha\left[R_i + \gamma_{a_i}^{\max} Q\left(S_{i+1}, a_t\right)\right] \quad (3)$$

where α is the learning rate, γ is the discount factor for descaling the current maximum *Q*-value for a row, S_i is the observed state at the decision epoch t_i, and S_{i+1} is the predicted state in the t_{i+1} -th decision epoch determined by the estimated workload and current performance (EWMA). The reward function R_i at t_i (4) is computed as a function of the resulting average slack ratio at the t_i-th decision epoch (*Li*) and its change since the last decision epoch (ΔL).

$$R_i = a\left|L_i\right| + b\Delta L \quad (4)$$

where *a* and *b* are predetermined constants to ensure actions improving L_i values are rewarded or vice-versa. The L_i values are estimated as:

$$L_i = \frac{1}{D\left(T_{ref}\right)} \sum_{t=0}^{n} \left(T_{ref} - T_i - T_{OVH}\right) \quad (5)$$

where T_{ref} is the reference execution time, T_i is the application task execution time, *D* is the number of elapsed decision epochs since the start of the application with a given T_{ref}, T_{OVH} is the total overheads caused by learning and adaptation steps (*V-F*). Hence, ΔL is calculated as the difference of L_{i-1} and L_i. The T_i can be calculated as the ratio of the observed processor cycles and the chosen operating frequency at the *i*-th decision epoch [12].

C. Exploitation

Following exploration, we have the exploitation phase where the learnt state–action relationships are exploited. The transition from the exploration to exploitation phase is greedy heuristic controlled through EP, denoted by ε ($0 \leq \varepsilon \leq 1$). To accelerate the process of exploitation, ε is updated as:

$$\varepsilon_i = \varepsilon_{i-1} \exp[-(1-\alpha)] \quad (6)$$

where α is the learning factor per decision epoch. Exploration or exploitation is carried out based on ε_i to determine the best explored *state-action* pair (in terms of positive reward).

D. Many-core formulation

The algorithmic formulation is adapted for many-core systems, through simple modifications. First, the predicted workload per core is normalized with respect to the total system workload as:

$$normCC_{i+1}^{j} = \frac{predCC_{i+1}^{j}}{\sum_{j}^{C} predCC_{i+1}^{j}} \quad (7)$$

where $predCC_{i+1}^{j}$ is the predicted workload and $normCC_{i+1}^{j}$ is the normalized workload, for the *j*-th core (*j*=1 to *C*, where *C* is the total number of cores on the embedded system). With a given discretised levels of normalized workload ($normCC_{i+1}^{j}$) and average slack ratio (*L*), a number of *Q*-table states are defined and organized in rows [similar to (2) and (3)]. For each state, the available *V-F* controls are used in the action space organized in 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 round-robin manner). Such *V-F* control per decision epoch helps to reduce *Q*-table complexity significantly as opposed to controlling multiple cores per decision epoch, which requires combinations of *V-F* settings of all cores in the *Q*-table.

III. Experimental Results

The adaptive energy minimization approach was implemented as a power governor in Linux kernel revision 3.10.96 running on the Odroid XU3 platform. In our experiments we use only the A15 cluster, supporting 19 *V-F* settings (2000 MHz – 200 MHz in 100 MHz steps). We test various applications: an MPEG4 decoder, FFT, and the PARSEC and SPLASH2 benchmarks. Each application is transformed to a periodic structure, where it is executed for several iterations each of which is accompanied by a deadline serving as the application's performance requirement. At each iteration, multiple threads are spawned with each thread performing a task on the input data. These iterations are termed frames in this context. Power is measured from on-board power sensors each frame and subsequently, the energy is calculated by multiplying average power with execution time.

A. Energy minimisation

An H.264-based video decoder application is executed with a football sequence of approximately 3000 frames using the following three approaches: 1) multicore DVFS control [20] (the thermal constraint was neglected for equivalence of comparison); 2) Linux on-demand governor per core [5]; 3) the approach in this paper. Table I highlights the normalized performance and energy consumption of the various approaches. It is important to note that performance is normalized to the required performance per frame (T_{ref}) and energy normalization is carried out with respect to Oracle (through offline determination of optimized *V-F* for the observed CPU workloads) performance. A normalised energy >1 implies higher energy consumption, while <1 is a lower consumption. Similarly, a normalised performance >1 implies underperformance while <1 is over-performance.

TABLE I
Comparative Evaluation of Normalised Energy and Performance Requirements [12]

Methodology	Normalized energy	Normalized performance
Linux Ondemand [5]	1.29	0.77
Multi-core DVFS control [20]	1.20	0.89
Proposed	1.11	0.96

The proposed approach achieves improved energy consumption compared to existing approaches. On-demand is agnostic of application performance requirements and hence consumes the most energy. Multi-core learning [20] and on-demand [5] over-perform due to poor adaptation to variations, resulting in up to 16% higher energy consumption.

B. Impact of state prediction

As a result of the EWMA algorithm, the *predicted* and *actual* workload (number of cycles, *CC*) and the average slack ratios (*L*) undergo mispredictions. This was observed while executing the MPEG4 decoding at 24 SVGA fps, as shown in Fig. 3. The smoothing factor (γ) for our analysis was experimentally determined as 0.6. There are mispredictions during the exploration frames for the first 25 frames and also during the exploitation phase after 90 frames. The highest average misprediction with respect to the average workload was approximately 8%, evident for the first 100 frames, with a lowest misprediction value of 3% following it. Under-prediction of the workload (where *actual* is higher than *predicted*) results in a deadline miss by the frames, whereas over-prediction (where *actual* is lower than *predicted*) results in higher power consumption. Most video decoders drop frames, which miss deadlines, resulting in a glitch in the output video which degrades user experience. The impact of state misprediction is mitigated by considering the current performance offset in terms of average slack (L_i) in conjunction with the current predicted workload ($predCC_i$) for mapping the next state (cf. Section II). In case of a performance offset (positive/negative L_i) caused through misprediction, the behavior is learnt and appropriate *V-F* control is applied to maximise reward (4).

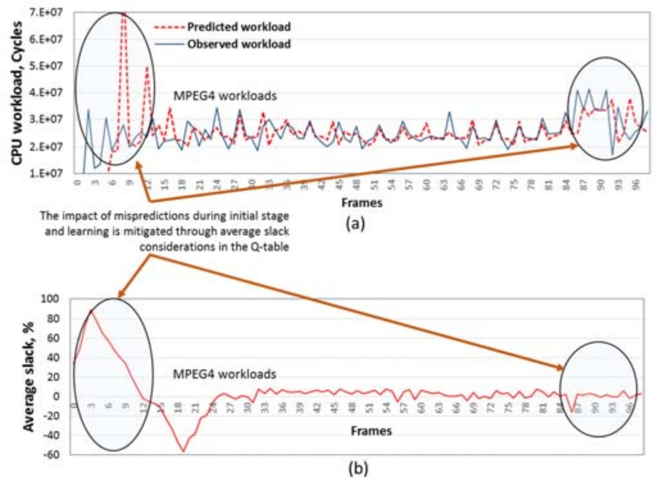

Fig. 3. Workload misprediction for MPEG4 and learning impact on average slack.

C. Number of Explorations

The advantage our approach employing EPD (2) during initial learning (i.e., RL step), is illustrated in Table II, highlighting the average number of explorations for three applications compared against an existing approach [21].

TABLE II
Comparative Evaluation of the Number of Explorations [12]

Application	Number of explorations	
	[21]	Our approach
MPEG4 (30 fps)	144	83
H.264 (15 fps)	149	90
FFT (32 fps)	119	74

It can be observed that our approach benefits from reduced exploration due to the relationship between current performance and the *V-F* action (4) [21]. FFT uses the fewest explorations since it exhibits less workload variations resulting in faster learning by the algorithm. MPEG4 and H.264 applications exhibited longer exploration due to higher workload variations and hence more states being visited.

D. Learning Overhead

The learning overhead has three components: (1) sensor sampling comprising performance counter register accesses, (2) processing and (3) *V-F* transitions. Processing and *V-F* transitions are key factors as they are compute intensive. The time overhead (T_{OVH}) was evaluated by averaging the differences of per-frame execution time of ffmpeg decoding three frames (T_{ref} – 31ms) compared to the multi-core DVFS control approach [20]. This is illustrated in Table III. As the learning of each core is shared by other cores in our approach, converges quicker and hence requires fewer decision epochs.

TABLE III
COMPARATIVE EVALUATION OF WORST CASE LEARNING OVERHEAD [12]

Methodology	**Time overhead (T_{OVH}) (in decision epochs)**
Multi-core DVFS control [20]	205
Our approach	105

IV. DISCUSSION

In this paper, we have proposed a run-time management approach for adaptive energy minimisation in multi-core embedded systems incurring low-overhead. The algorithm uses Q-learning to select *V-F* control for a predicted workload and a given application performance requirement. The algorithm is biologically inspired from the manner a human brain reacts and adapts to a changing environment to maximise rewards and reinforce optimal decisions, over a long period. The proposed RTM is implemented as a power governor, and validated through experiments with real applications and various benchmark suites, achieving up to 16% energy savings compared to state-of-the art. Our future work is investigating how to extend this approach to manage the energy consumption of multiple concurrently executing applications.

REFERENCES

[1] D.N. Truong, W.H. Cheng, T. Mohsenin, et al. "A 167-processor computational platform in 65 nm CMOS," IEEE Journal of Solid-State Circuits 44, no. 4 (2009): 1130-1144.

[2] F. David. "An ARM perspective on addressing low-power energy-efficient SoC designs," Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design, ACM, 2012.

[3] K. Choi, W.C. Cheng, and M. Pedram. "Frame-based dynamic voltage and frequency scaling for an MPEG player," Journal of Low Power Electronics 1.1 (2005): 27-43.

[4] Y. Gu, and S. Chakraborty, "Control theory-based DVS for interactive 3D games," in Proceedings of the 45th annual Design Automation Conference, 2008, pp. 740–745.

[5] V. Pallipadi, and A. Starikovskiy, "The ondemand governor," in Proceedings of the Linux Symposium, 2006, vol. 2, pp. 215–230.

[6] M. Pedram, "Power optimization and management in embedded systems," Proceedings of the 2001 Asia and South Pacific Design Automation Conference, ACM, 2001.

[7] R. Jejurikar, and R. Gupta, "Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems," Low Power Electronics and Design, 2004. ISLPED'04. Proceedings of the 2004 International Symposium on. IEEE, 2004.

[8] S. Yue, D. Zhu, Y.Wang, and M. Pedram, "Reinforcement learning based dynamic power management with a hybrid power supply," in Proc. IEEE 30th Int. Conf. Comput. Design (ICCD), Montreal, QC, Canada, 2012, pp. 81–86.

[9] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," ACM Trans. Design Autom. Electron. Syst., vol. 18, no. 2, pp. 1–32, Mar. 2013.

[10] R. Ye, Q. Xu, "Learning-based power management for multicore processors via idle period manipulation," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 33.7 (2014): 1043-1055.

[11] A. Das, B.M. Al-Hashimi, and G.V. Merrett, "Adaptive and Hierarchical Runtime Manager for Energy-Aware Thermal Management of Embedded Systems." ACM Transactions on Embedded Computing Systems (TECS) 15.2 (2016): 24.

[12] R.A. Shafik, S. Yang, A. Das, L.A. Maeda-Nunez, G.V. Merrett and B.M. Al-Hashimi, "Learning Transfer-Based Adaptive Energy Minimization in Embedded Systems," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 6, pp. 877-890, June 2016.

[13] P. Tommasino, D. Caligiore, M. Mirolli, and G. Baldassarre, "A Reinforcement Learning Architecture that Transfers Knowledge between Skills when Solving Multiple Tasks," IEEE Transactions on Cognitive and Developmental Systems (2016).

[14] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic knobs for responsive power-aware computing," in ACM SIGPLAN Notices, 2011, vol. 46, no. 3, pp. 199–212.

[15] D. Silver, Reinforcement Learning Advanced Topics 2015 (COMPM050/COMPGI13), UCL lecture notes. Available: www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html [Accessed: Jan 16]

[16] A. Sinha, and A.P. Chandrakasan, "Dynamic voltage scheduling using adaptive filtering of workload traces," In VLSI Design, 2001. Fourteenth International Conference on, pp. 221-226. IEEE, 2001.

[17] S. Sinha, S.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 1, no. 3 (2011): 381-390.

[18] A.K. Coskun, T.S. Rosing, and K.C. Gross, "Utilizing predictors for efficient thermal management in multiprocessor SoCs," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 28.10 (2009): 1503-1516.

[19] T. Jiang, D. Grace, and P. D. Mitchell, "Efficient exploration in reinforcement learning-based cognitive radio spectrum sharing," IET Commun., vol. 5, no. 10, pp. 1309–1317, Jul. 2011.

[20] Y. Ge, and Q Qiu, "Dynamic thermal management for multimedia applications using machine learning," Proceedings of the 48th Design Automation Conference, ACM, 2011.

[21] 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 (TODAES) 18, no. 2 (2013): 24.