HiLITE: Hierarchical and Lightweight Imitation Learning for Power Management of Embedded SoCs

Anderson L. Sartor, Anish Krishnakumar, Samet E. Arda, Umit Y. Ogras, and Radu Marculescu

Abstract—Modern systems-on-chip (SoCs) use dynamic power management (DPM) techniques to improve energy efficiency. However, existing techniques are unable to efficiently adapt the runtime decisions considering multiple objectives (e.g., energy and real-time requirements) simultaneously on heterogeneous platforms. To address this need, we propose HiLITE, a hierarchical imitation learning framework that maximizes the energy efficiency while satisfying soft real-time constraints on embedded SoCs. Our approach first trains DPM policies using imitation learning; then, it applies a regression policy at runtime to minimize deadline misses. HiLITE improves the energy-delay product by 40 percent on average, and reduces deadline misses by up to 76 percent, compared to state-of-the-art approaches. In addition, we show that the trained policies not only achieve high accuracy, but also have negligible prediction time overhead and small memory footprint.

Index Terms—Hierarchical imitation learning, dynamic power management, machine learning, real-time, SoC

1 Introduction

Systems-on-Chip (SoCs) should be designed to meet aggressive performance requirements while coping with limited battery capacity, thermal design power (TDP), and real-time (RT) constraints. A step in this direction consists of exploiting heterogeneity, e.g., using big cores when high performance is needed and switching to little cores otherwise. In addition, techniques such as dynamic voltage and frequency scaling (DVFS) and power gating (PG) can be used at runtime to manage the power consumption of SoCs. However, the design space of runtime decisions explodes combinatorially with the number of cores, frequency levels, and power states. Additionally, current platforms serve a wide range of applications with distinct characteristics and requirements. The extensive design space and the growing variety of applications call for new runtime techniques to efficiently manage the power and performance of embedded heterogeneous platforms.

Prior work on heterogeneous platforms (e.g., [1], [7], [10]) use machine learning to improve the energy efficiency w.r.t. dynamic power management (DPM) techniques present in commercial SoCs. However, these studies do not take RT constraints and PG into consideration. Likewise, hierarchical power management techniques like [5] and [6] do not target these metrics; instead they use reinforcement learning (RL) and specialized heuristics for energy optimization in homogeneous platforms. Targeting additional constraints such as real-time is non-trivial, and if the DPM techniques do not apply specific mechanisms to address these constraints, they deliver suboptimal results. For instance, such an approach results in high deadline misses for RT applications, which we also evaluate in Section 3.

The authors in [8] and [9] use RL to optimize for RT constraints. However, RL increases exponentially in size as the state and action spaces increase. Instead, we use imitation learning (IL) to train a DPM policy that efficiently explores a large design space. While HyPowMan [3] considers RT and PG, their approach simply selects between two heuristic-based policies for DVFS and PG. These three previous approaches only consider homogeneous platforms and often single-core processors, hence, such techniques are not able to efficiently optimize a heterogeneous SoC. Therefore, given the constraints in current SoCs, new techniques that are able to adapt the runtime decisions to different objectives and constraints are needed.

In this paper, we propose HiLITE, a hierarchical DPM framework that uses IL to minimize the energy-delay product (EDP), while coping with soft RT constraints in SoCs. To this end, we first construct an oracle using power and performance data of domain-specific applications, namely wireless communications and radar systems. We then train IL policies to achieve low EDP while considering soft deadlines, by adjusting the frequency and number of active cores in LITTLE and big clusters. We note that an offline trained policy can set the operating point successfully for energy optimization, but it may miss deadlines due to the unpredictable dynamic variations of the workload and scheduling. Therefore, we also propose *a novel online regression policy* that fine-tunes the policy decisions to address these variations. The main contributions of this paper are as follows:

- A hierarchical framework that comprises lightweight IL policies to maximize energy-efficiency and a regression policy for fine-tuning the SoC configuration to meet RT constraints.
- Design- and run-time approaches for coping with execution deadlines, while optimizing the energy consumption.
- Validation of the simulation results against a commercial SoC w.r.t. performance, power, and temperature.

2 Hierarchical Deadline-Aware DPM

This section presents oracle generation methodology and deadline-aware IL policies in HiLITE, as illustrated in Fig. 1.

2.1 Oracle Generation

To characterize the impact of the power management configuration (e.g., cluster frequencies) on system performance and energy consumption and enable Oracle generation, we construct *microbenchmarks* that consist of a fixed number of frames. The frames are the basic unit of data processed by each application, so each frame contains 64 bits for most target applications. We run each microbenchmark on the Odroid-XU3 board for each supported configuration and store performance counters, execution time, and power consumption. Therefore, this methodology preserves the workload when evaluating microbenchmarks with different frequency levels and number of cores [7]. In our experiments, each microbenchmark consists of 10 frames and is long enough to collect reliable statistics. The frames within each microbenchmark are executed in parallel based on the availability of resources and the rate at which they are injected into the system. Finally, a *workload* is a collection of such microbenchmarks.

We evaluate all possible combinations for 10 frames in a microbenchmark with five applications (presented in Section 3.2), resulting in 1,001 unique microbenchmarks. Each of these microbenchmarks

- *A.L. Sartor is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213. E-mail: asartor@cmu.edu.*
- *A. Krishnakumar, S.E. Arda, and U.Y. Ogras are with the School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85281. E-mail: {anishnk, sarda1, umit}@asu.edu.*
- *R. Marculescu is with the Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712. E-mail: radum@utexas.edu.*

Manuscript received 22 Mar. 2020; revised 23 Apr. 2020; accepted 30 Apr. 2020. Date of publication 4 May 2020; date of current version 29 May 2020.
(Corresponding author: Anderson L. Sartor.)
Digital Object Identifier no. 10.1109/LCA.2020.2992182

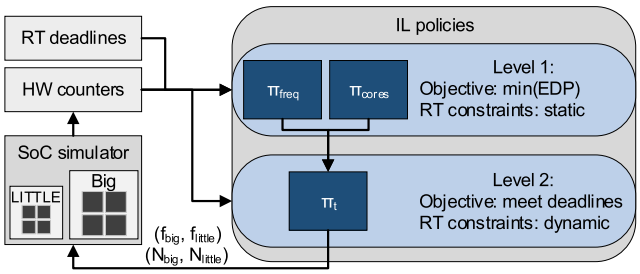

Fig. 1. Overview of HiLITE. Level 1 contains IL policies for predicting the ideal frequency (π_{freq}) and number of cores (π_{cores}) to minimize EDP (Section 2.2), taking the deadlines into account during the oracle generation (Section 2.1). Level 2 dynamically adjusts level 1 decisions based on a regression policy (π_t) that estimates the execution time to meet the deadlines (Section 2.3).

is evaluated with all combinations of frequency states: 8 for the big cluster (0.6-2.0 GHz) and 5 for the LITTLE (0.6-1.4 GHz), using a 200 MHz step, and all number of cores (1-4 big and 1-4 LITTLE); this results in more than 640 K samples. Based on these samples, the oracle chooses the configuration that minimizes the EDP of each microbenchmark, while considering the RT constraints. The *oracle* (π_k^*) for each microbenchmark k is expressed as

$$\pi_k^* = \arg\min_{C_i \in C} \quad \{E_k(C_i) \times t_k(C_i)\}$$
$$\text{s.t.} \quad t_k(C_i) \leq D_k, \qquad k = 1, 2, \ldots, \binom{F+A-1}{A-1}, \tag{1}$$

where C represents all possible combinations of frequency points and number of cores (f_{points}(LITTLE) $\times$ f_{points}(big) $\times$ N_{cores}(LITTLE) $\times$ N_{cores}(big)). The index k represents each unique microbenchmark based on the binomial coefficient of F frames per microbenchmark and A applications. $E_k(C_i)$ and $t_k(C_i)$ denote the energy consumption and execution time of a given microbenchmark at configuration C_i, respectively. Finally, D_k is the deadline of microbenchmark k.

The oracle generates two tuples in the following format: (f_{big}, f_{little}), (N_{big}, N_{little}), for the policies predicting the frequency and number of cores, respectively. Similarly, for the regression policy, the measured execution time is used as the oracle.

2.2 Imitation Learning Policies

Exact imitation trains a policy that closely follows the oracle. However, it may suffer from error propagation, i.e., errors in previous states affect the decisions for the next ones. To address this issue, we employ an IL approach called DAgger [11], which is applied to all three proposed policies (prediction of frequencies, number of cores, and execution time). More precisely, at every control interval (typically 50-100 ms [4]), the policy makes a prediction, which is applied to the system and compared against the oracle. If the prediction differs from the oracle, this sample is aggregated to the dataset and the execution continues. No action is taken if the prediction aligns with the oracle. After the execution finishes, the aggregated dataset is used to retrain the policy, in order to teach the policy to learn from the mistakes made during the previous iterations.

Table 1 presents the features that are used to train the policies. These hardware counters are normalized to the number of instructions in order to generalize to other applications with similar characteristics.

2.3 Runtime Management of Deadline Constraints

We propose both design- and run-time techniques to cope with deadline constraints. More precisely, at the design time, we modify the oracle generation to consider RT-constraints, as in Equation (1).

TABLE 1
Features for Training the IL Policies

Current state	Number active cores (big cluster)	HW counters	CPU cycles
	Number active cores (LITTLE cluster)		Branch misprediction
	Frequency big cluster		L2 cache misses
	Frequency LITTLE cluster		Data memory access
			Non-cache external mem. req.

At runtime, we employ a hierarchical approach to estimate the execution time, which addresses the workload runtime variations.

Algorithm 1. Hierarchical Structure of HiLITE

```
   /*Level 1*/
 1 s ← get current state and hardware counters
 2 foreach Cl_i in Clusters do
 3   f(Cl_i) ← π_freq(s)[Cl_i]
 4   V(Cl_i) ← voltage point w.r.t. f(Cl_i)
 5   cores(Cl_i) ← π_cores(s)[Cl_i]
   /*Level 2*/
 6 if workload has real-time constraints then
 7   s ← get current state and hardware counters
 8   t_pred(k) ← π_t(s)
 9   if t_pred(k) > D_k then
10     foreach Cl_i in Clusters do
11       if f(Cl_i) < max{f(Cl_i)} then
12         f(Cl_i) ← next frequency point of Cl_i
13           V(Cl_i) ← next voltage point of Cl_i
14     if ∀Cl_i in Clusters, f(Cl_i) = max{f(Cl_i)} then
15       foreach Cl_i in Clusters do
16         if cores(Cl_i) < max{cores(Cl_i)} then
17           cores(Cl_i) ← cores(Cl_i) + 1
18 ∀Cl_i in Clusters, apply PG to the inactive cores of Cl_i
```

Algorithm 1 depicts the proposed hierarchical structure of HiLITE; this algorithm is applied at each control interval. First, the IL policies get the current system state and hardware counters, then perform the inference for the frequency and number of cores. The regression policy is activated only if the microbenchmark has a deadline. Then, the predicted execution time t_{pred} (line 8 in Algorithm 1) is compared against the deadline; if the predicted execution time is greater than the deadline D_k, the following measures are applied: 1) increase the frequency/voltage of the big and LITTLE clusters by one increment (lines 12-13), and 2) if the frequency is already at the maximum for both clusters, increase the number of active cores by one (line 17). Otherwise, if the predicted time does not exceed the deadline, the execution continues following the level 1 policies. PG is applied to the inactive cores of each cluster. We choose to increase the frequency first because turning on an additional core incurs more overhead than increasing the frequency of an active core [10].

As the hierarchical approach is applied at runtime, the frequency and number of cores are fine-tuned only if necessary. If this methodology is applied entirely at design-time, the oracle decisions overestimate the required frequency and number of cores for all samples, to achieve the same level of deadline misses from the hierarchical approach. This increases the EDP by around 20 percent w.r.t. the hierarchical policies. Hence, the hierarchical approach addresses these issues by providing a generic and more efficient solution at runtime.

3 Experimental Results

3.1 Methodology and Experimental Setup

Since we are focusing on lightweight IL techniques, we use decision trees for level 1 and a regression tree for level 2 to achieve fast training and inference. *For training*, we use leave-one-out cross-

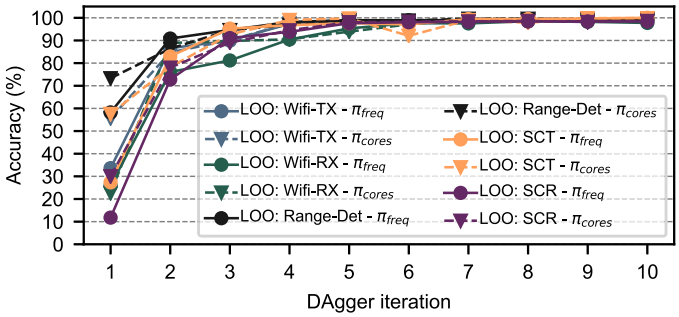

Fig. 2. Decision tree accuracy as DAgger iteratively trains the policies. Leave-One-Out (LOO) cross-validation is used for each application.

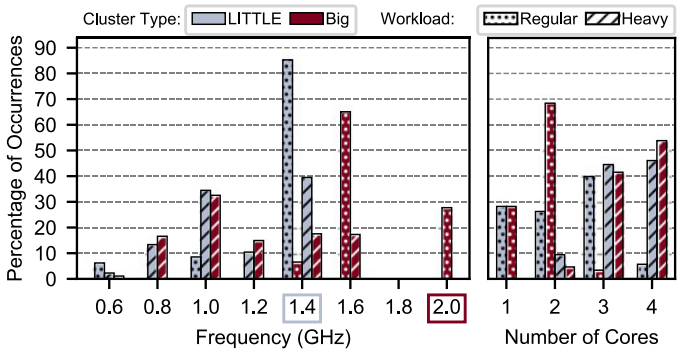

Fig. 3. Normalized histogram w.r.t. the total predictions for frequency (π_{freq}) and number of cores (π_{cores}). The highlighted frequencies depict the maximum frequency for LITTLE (1.4 GHz) and big (2 GHz) clusters. Each bar represents a combination of cluster type and workload, i.e., the LITTLE and big clusters are depicted in different colors, while the regular and heavy workloads are depicted with different patterns.

validation to completely remove frames from a specific application from the dataset. Then, we run a workload that contains frames from the removed application to test the model generalization to unseen applications. *For testing*, we consider workloads with 50 microbenchmarks and execute such workloads 5 times (standard deviation of less than 1 percent). For each execution, we apply 10 DAgger iterations.

Two main scenarios are evaluated with the target applications: 1) *Regular workload* of a communication system, having an average of 1.25 frames being processed in parallel with up to 5 parallel frames; and 2) *Heavy workload* with 3 parallel frames on average and up to 8 frames (i.e., 100 percent utilization as we have 8 cores in total).

We test the proposed approach under different RT-constrained scenarios by generating bounded random deadlines (D_k) for each microbenchmark. This allows the generation of deadline constraints based on profiled requirements instead of manually inputting the deadline for each microbenchmark, hence, allowing a flexible evaluation of different scenarios. To achieve this, we generate a random number R from a uniform distribution U between a specified low (D_{TLow}) and high (D_{THigh}) thresholds. These thresholds can range from 0 to 100 percent. Then, we multiply R by the range of the microbenchmark's execution time and add the minimum execution time $\min_{C_i \in C}\{t_k(C_i)\}$. So, the deadline for microbenchmark k is given by:

$$D_k = \min_{C_i \in C}\{t_k(C_i)\} + (R \sim U([D_{TLow}, D_{THigh}]) \div 100) \times (k = \max_{C_i \in C}\{t_k(C_i)\} - k = \min_{C_i \in C}\{t_k(C_i)\}). \quad (2)$$

We evaluate the following deadline ranges: D_T = 0-5%, D_T = 5-10%, and D_T = 10-20%, in decreasing order of difficulty to satisfy.

3.2 Simulation Framework Overview and Validation

We extended the system-level SoC simulator proposed in [2] to incorporate the proposed IL technique.

Platform Model. To ensure high fidelity, the simulator is calibrated using the performance monitoring unit (PMU), current and temperature sensors of the Odroid-XU3. This board allows changing the frequencies only at the cluster level and does not apply PG. To make the design more flexible and explore better power/performance trade-offs, we have implemented a per-core PG technique in the simulator.

Benchmark Applications. We consider five multi-threaded reference applications from wireless communications and radar processing domains: WiFi transmitter (WiFi-TX), WiFi receiver (WiFi-RX), range detection (Range-Det), single-carrier transmitter (SCT), and single-carrier receiver (SCR). All these are representative examples of streaming applications with soft RT constraints.

*Data Collection.*We instrument the applications' source code with performance application programming interface (PAPI) calls to profile power, performance, and hardware counter data on the Odroid-XU3.

We compare execution time, power, and temperature reported by our simulator against measurements on an Odroid-XU3, while running the benchmark applications. The simulator has only 2.8, 6.1, and 2.4 percent error, on average, for these three metrics when we sweep the LITTLE and big core frequencies for multi-threaded applications. Similarly, when we sweep the different number of cores, we have 2.7, 1.3, and 3.8 percent error on average, respectively. The complete evaluation for both single- and multi-threaded applications and the validation data can be found in [2].

3.3 Accuracy Evaluation

The decision tree quickly learns as we apply DAgger iterations (Fig. 2), and achieves 99.1 percent accuracy on average. The accuracy at the first iteration ranges from 11.8 to 73.3 percent, and by the fourth iteration, all policies are already above 90 percent. The regression policy (level 2) achieves an R^2 metric of 99.7 percent, closely following the oracle. The policies take from 0.013 to 0.617 ms per prediction, which is negligible over 50-100 ms control epochs. Likewise, the memory requirements range from 3 to 280 KB.

Next, we analyze the predictions w.r.t. frequency (π_{freq}) and number of cores (π_{cores}) made by the IL policies, presented in Fig. 3. When running a regular load, π_{freq} chooses 85 percent of the time the maximum frequency for the LITTLE (i.e., 1.4 GHz), and high frequencies for the big cluster (65 percent at 1.6 GHz and 28 percent at 2 GHz). At the same time, π_{cores} chooses more than 95 percent of the time three or fewer LITTLE cores and two or fewer big cores. This shows that the policies effectively shut down cores when the workload is not heavy. For heavy workloads, the frequencies lie mostly within 0.8 to 1.6 GHz, and π_{cores} chooses mostly 3 and 4 cores (around 90 percent of the time), such that the EDP is minimized as we have several frames being processed in parallel.

3.4 Comparison With State-of-the-Art Techniques

We compare our approach against *performance*, *powersave*, *ondemand*, and DyPO [7]. The first three belong to the Linux governors and the latter uses machine learning to adjust frequency and number of cores. We evaluate two versions of HiLITE: First, only changing the frequency of the clusters (HiLITE$_{(F)}$), and second, by changing both frequency and number of cores (HiLITE$_{(F-C)}$).

EDP Evaluation. HiLITE$_{(F)}$ achieves 2 and 29 percent reduction for regular and heavy workload scenarios, respectively, as shown in Fig. 4a. The former case leads to smaller improvement since the

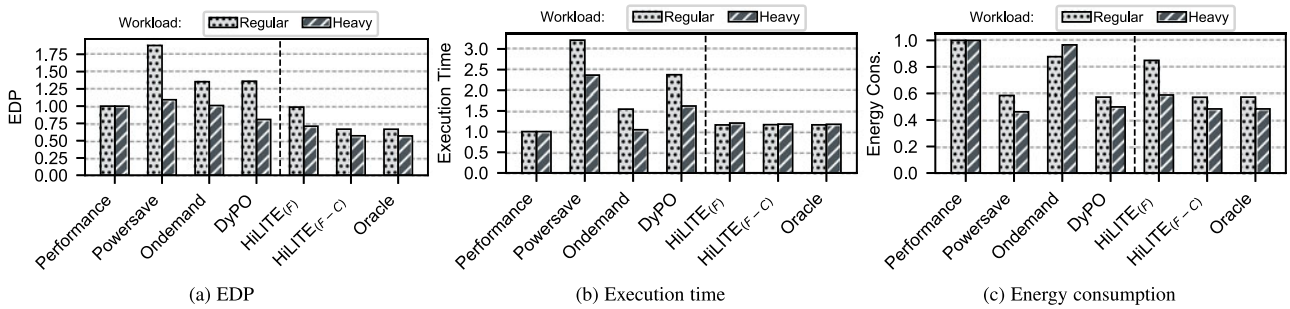

Fig. 4. Normalized EDP, execution time, and energy consumption w.r.t. *performance* governor. HiLITE$_{(F)}$ represents the IL policy that adjusts the frequency, and HiLITE$_{(F-C)}$, the policy for prediction of both frequency and number of cores. The lower the values, the better.

oracle frequency is high for most of the execution, as discussed in Fig. 3. When we apply HiLITE$_{(F-C)}$, the EDP improvement rises to 34 and 43 percent w.r.t. the *performance* governor, for the same runtime scenarios. Compared to DyPO, HiLITE$_{(F-C)}$ achieves 51 and 29 percent lower EDP under regular and heavy workloads, respectively. We note that DyPO is not able to efficiently explore such a large design space since it employs logistic regression followed by k-means clustering; also, DyPO does not exploit PG to further improve the energy efficiency. In addition, HiLITE$_{(F-C)}$ closely follows the Oracle, being within 0.4 percent of the Oracle's mark, which is the upper bound for comparison.

Execution Time Evaluation. HiLITE achieves low performance degradation w.r.t. the *performance* mode (16-21 percent), depicted in Fig. 4b. While the other baselines have considerably higher degradation: *powersave* 136-221 percent, *ondemand* 5-54 percent, and DyPO 61-137 percent.

Energy Consumption Evaluation. HiLITE$_{(F-C)}$ achieves 43 and 52 percent energy savings w.r.t. the *performance* mode under a regular and heavy workload, respectively, as shown in Fig. 4c.

3.5 Evaluations Under Real-Time Constraints

Fig. 5 presents the normalized EDP and the percentage of missed deadlines for different threshold ranges. As DyPO has considerably higher performance degradation than HiLITE$_{(F-C)}$ (this leads to even higher deadline misses), we choose HiLITE$_{(F-C)}$ as the baseline. HiLITE$_{(RT)}$ represents HiLITE with RT optimization enabled, and under D_T = 0-5% (i.e., tightest deadlines); this reduces the deadline misses from 87 to 11 percent, and from 88 to 40 percent for regular and heavy workloads, respectively. The reduction in the latter case is lower due to multiple frames being processed in parallel. The same trend is observed for D_T = 5-10%, for the regular workload (70 percent deadline misses are reduced to 0 percent) and for the heavy workload (52 to 17 percent). Further relaxation of these deadline constraints drives the missed deadlines towards zero for both workloads.

For regular workloads, the EDP overhead is low, only 2 percent on average, while under a heavy workload, there is a trade-off between minimizing the EDP and meeting the deadlines. If the deadlines are prioritized, the EDP improvement goes from 43 to 25 percent, as the frequency needs to be increased to meet the deadlines.

This evaluation shows that generating an oracle without RT information leads to a high number of deadline misses (close to 90 percent). In contrast, our proposed approach adds the RT information to the oracle generation and uses a dynamic regression policy to address the runtime variation in the execution time.

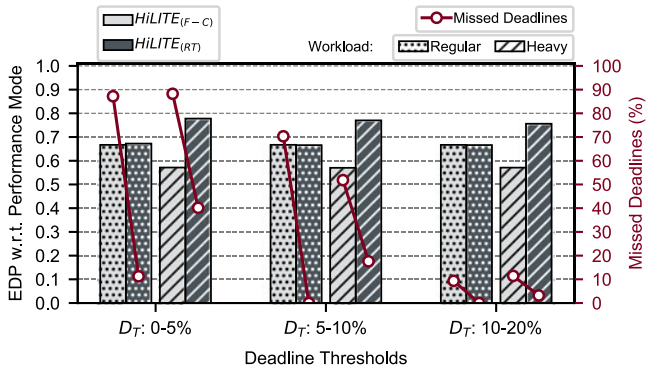

Fig. 5. Normalized EDP and percentage of missed deadlines for different techniques and deadline thresholds. The lower the threshold, the closer the deadline is to the minimum execution time.

4 Conclusion and Future Work

In this work, we have proposed a hierarchical and lightweight IL power management framework (HiLITE) for improving the energy efficiency of embedded SoCs (achieving around 40 percent EDP reduction w.r.t. state-of-the-art), while taking into consideration soft RT constraints and reducing deadline misses by up to 76 percent. As future work, HiLITE will also be applied to hardware accelerators and integrated with energy-aware schedulers.

Acknowledgments

This material is based on research sponsored by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-2-7960. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusion contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) or the U.S. Government.

References

[1] A. Aalsaud *et al.*, "Power–aware performance adaptation of concurrent applications in heterogeneous many-core systems," in *Proc. Int. Symp. Low Power Electron. Design*, 2016, pp. 368–373.

[2] S. Arda *et al.*, "DS3: A system-level domain-specific system-on-chip simulation framework," *IEEE Trans. Comput.*, to be published, doi: 10.1109/TC.2020.2986963.

[3] K. Bhatti, C. Belleudy, and M. Auguin, "Power management in real time embedded systems through online and adaptive interplay of DPM and DVFS policies," in *Proc. IEEE/IFIP Int. Conf. Embedded Ubiquitous Comput.*, 2010, pp. 184–191.

[4] D. Brodowski, N. Golde, R. J. Wysocki, and V. Kumar, "Linux CPUFreq governors." [Online]. Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[5] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *Proc. Design Autom. Test Eur. Conf.*, 2015, pp. 1521–1526.

[6] A. Das, B. M. Al-Hashimi, and G. V. Merrett, "Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 15, no. 2, pp. 24:1–24:25, Jan. 2016.
[7] U. Gupta *et al.*, "DyPO: Dynamic pareto-optimal configuration selection for heterogeneous MpSoCs," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, pp. 123:1–123:20, Sep. 2017.
[8] F. M. M. Islam and M. Lin, "Hybrid DVFS scheduling for real-time systems based on reinforcement learning," *IEEE Syst. J.*, vol. 11, no. 2, pp. 931–940, Jun. 2017.
[9] F. M. M. Islam, M. Lin, L. T. Yang, and K.-K. R. Choo, "Task aware hybrid DVFS for multi-core real-time systems using machine learning," *Inf. Sci.*, vol. 433/434, pp. 315–332, 2018.
[10] S. K. Mandal, G. Bhat, C. A. Patil, J. R. Doppa, P. P. Pande, and U. Y. Ogras, "Dynamic resource management of heterogeneous mobile platforms via imitation learning," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 12, pp. 2842–2854, Dec. 2019.
[11] S. Ross, G. Gordon, and J. A. D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. Int. Conf. Artif. Intell. Stat.*, 2011, pp. 627–635.