Improve the Stability and Robustness of Power Management through Model-free Deep Reinforcement Learning

Lin Chen[1], Xiao Li[1], Jiang Xu[2,1,†]
[1]Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology
[2]Microelectronics Thrust, The Hong Kong University of Science and Technology
[†]*Corresponding author: jiang.xu@ust.hk*

***Abstract*—Achieving high performance with low energy consumption has become a primary design objective in multi-core systems. Recently, power management based on reinforcement learning has shown great potential in adapting to dynamic environments without much prior knowledge. However, conventional Q-learning (QL) algorithms adopted in most existing works encounter serious problems about scalability, instability, and overestimation. In this paper, we present a deep reinforcement learning-based approach to improve the stability and robustness of power management while reducing the energy-delay product (EDP) under user-specified performance requirements. The comprehensive status of the system is monitored periodically, making our controller sensitive to environmental change. To further improve the learning effectiveness, knowledge sharing among multiple devices is implemented in our approach. Experimental results on multiple realistic applications show that the proposed method can reduce the instability up to 68% compared with QL. Through knowledge sharing among multiple devices, our federated approach achieves around 4.8% EDP improvement over QL on average.**

***Index Terms*—power management, deep reinforcement learning, experience replay, federated learning, multicore system**

I. Introduction

Energy consumption has become a non-negligible problem in recent years, especially in mobile devices. High energy consumption shortens the operation time of battery-powered products and degrades the reliability of the whole system, creating the demand for thermal management. To release the mismatch between the limited battery and users' demand for high performance, efficient power management is indispensable in modern systems. Among various components, the processor is one of the most dominant energy consumers [1], [2]; therefore, improving the energy efficiency of the processor is a key target in power management.

Dynamic voltage and frequency scaling (DVFS) is one of the most widely used techniques in power management, and it dynamically changes the V/F levels of cores according to the runtime status of processors. The uncertainty and diversity of the environment and applications make it difficult for designers to implement a robust and adaptive policy based on heuristic methods [3], [4]. As a result, reinforcement learning (RL)-based DVFS control [5]–[9] has been proposed and shows significant improvement without much prior knowledge of the applications. However, Q-learning, one of the most popular and widely used RL algorithms, performs poorly in some stochastic environments due to large overestimations [10]. Additionally, to discretize state parameters, some prior knowledge is still required, especially when the partition is not even. Improper state partition may cause failed learning or instability during the converging process, while a fine partition maintaining a tremendous Q-table leads to a great increase in memory overhead and convergence time. Recently, the stability of online RL methods has become a major concern in safety-related and potentially hazardous applications [11]. The ability to recover from exploratory actions and significant environmental change should also be considered in a robust RL approach [12]. In the RL-based power management area, most works only focus on the quality of learned results [5]–[9], e.g., energy consumption reduction, ignoring the stability and recovering ability during the online process. However, these factors are highly related to the reliability of the whole system.

To overcome the scalability and instability obstacles in Q-learning and improve the effectiveness of power management, deep reinforcement learning (DRL) using value approximation is adopted in our method. We monitor multiple state features and update the policy periodically. In this way, runtime behaviors and unpredictable changes can be observed adequately and timely; hence, high robustness can be achieved. Nonetheless, there still exists one major limitation in DRL. Due to continuous state space and high variability from applications, it is hard for DRL agents to explore sufficiently, resulting in overfitting to some applications and a long convergence time. Computation overhead for both online and offline learning can not be ignored while high accuracy or performance is pursued. Consequently, we introduce federated learning allowing knowledge sharing among devices to further improve energy efficiency. Each device owns a local DQN-learner to interact with its local environment and shares its cumulated knowledge with others. Modern mobile devices mostly have easy access to WiFi and LTE networks, thus, knowledge sharing is available and realizable in our model. Federating learning can accelerate the learning speed and improve the quality of the policy, which is highly valuable in the learning-based power management [7]. The main contributions of our work are as follows:

- We propose a deep Q-networks (DQN)-based DVFS control approach to reduce the EDP of multi-core systems in a stable and robust way.
- We design appropriate state space and adopt periodical training to enhancing stability and robustness.
- We propose a federated DQN (FDQN)-based DVFS control approach using knowledge sharing.
- We define an online instability to measure and analyze the power-related stability.
- We quantitatively evaluate the efficiency of our proposed methods compared with some popular baseline methods.

978-3-9819263-6-1/DATE22/©2022 EDAA

II. DVFS Control based on Deep Q-networks

A. Deep Q-networks Basics

Q-learning (QL) is one of the most popular reinforcement learning (RL) algorithms. At each learning epoch t, Q-values are updated as

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a')]. \quad (1)$$

In Eq. (1), $Q(s_t, a_t)$ is updated after the QL agent experiences (s_t, a_t), transits to state s_{t+1}, and receives reward r_t. α denotes learning rate, and γ denotes the discount factor.

Deep Q-networks (DQN) is a popular algorithm proposed to resolve the curse of dimensionality in RL. Neural networks approximate Q-values through experience replay. The loss function of the networks is defined as

$$L(\theta) = \mathbb{E}[(r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta) - Q(s_t, a_t; \theta))^2]. \quad (2)$$

In DQN, continuous state parameters are directly input to the networks without discrete partition. Memory overhead of QL for maintaining a huge Q-table can be released. Additionally, due to the complexity of the value function derived by neural works, unseen relationships among state transitions and unvisited state space can be predicted with high accuracy; thus, DQN can achieve a more efficient policy.

B. Definition of Instability

Stability awareness and analyses of adaptive control methods are crucial especially in online algorithms [11]–[13]. In our work, we define an online instability I as the coefficient of variation (CV) of energy consumption during the learning process. I is calculated as

$$I(T) = \frac{\sqrt{\frac{1}{T}\sum_{t=1}^{T}(e_t - \bar{e})^2}}{\bar{e}}. \quad (3)$$

where e_t is energy consumption at the t-th step and $\bar{e}$ is average energy consumption of T steps.

We use CV to exclude the effect of energy value. To maintain stable energy usage, we expect the instability of several consecutive iterations generated by the learner is low. Additionally, in the early stage, we should also consider the ability to recover from unsuccessful exploration known as asymptotic stability in control theory.

C. DVFS Control based on Deep Q-networks

An overview of the DQN-based DVFS control is shown in Fig. 1. The DQN-learner acts as a DVFS controller for a multi-core processor outputting a set of V/F levels for all the cores according to their current states. Meanwhile, the DQN-learner updates its policy using samples in a replay buffer, which stores transitions (s_t, a_t, r_t, s_{t+1}) from the cores. The objective of our DQN-learner is to reduce the processor's EDP while satisfying user-defined performance requirements, e.g., 5% or 10% boundary of performance loss.

As shown in Table I, we use four features to construct our state space S. These hardware-level features reflect different aspects of the runtime status of the cores. Compared to the state

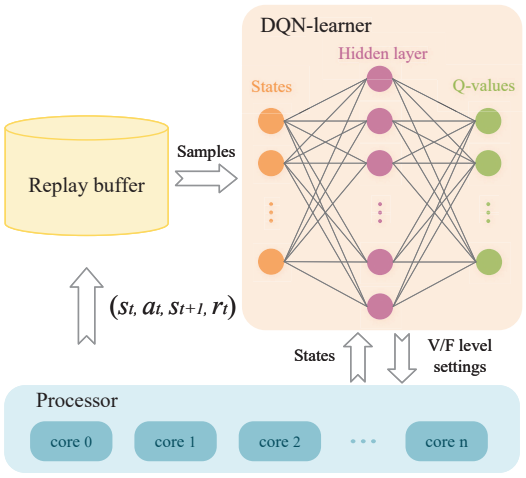


Fig. 1. Overview of DQN-based DVFS control.

space with a single feature, our comprehensive state description allows our learner to fully understand the complex system and catch up with dynamic changes timely. Action space A is matched with available V/F levels of the DVFS module. As for reinforcement signal R, both energy and performance are considered reward measurements to achieve a low EDP.

Here, cores in one processor are monitored and controlled separately. One DVFS control unit is called one epoch. In epoch t for core i, the experience $(s^i_{t-1}, a^i_{t-1}, r^i_{t-1}, s^i_t)$ is recorded and action a^i_t is decided according to the current policy and current state s^i_t. Design details of the DQN-based approach are interpreted as follows.

Busy time ratio (BTR) reflects the utilization degree of cores. A high BTR means the core is highly utilized at that time. BTR is calculated as

$$BTR_t = \frac{T^t_{busy}}{T^t_{epoch}}, \quad (4)$$

where T_{busy} excludes idle time from epoch time T_{epoch}.

Instruction per busy cycle ($IPBC$) is calculated as

$$IPBC_t = \frac{IC_t}{T^t_{busy} * f_t}, \quad (5)$$

where IC_t is the executed instruction count and f_t is the working frequency. $IPBC$ concentrates on instruction execution speed during busy cycles.

Performance loss (PL) is defined to represent performance degradation when cores work at a low V/F level with respect to the highest V/F level. The loss in one epoch can be measured by the discounted instruction count and transformed to the time dimension thereafter. The calculation equation is shown as follows:

$$PL_t = \frac{IC_{max} - IC_t}{IC_{max}} = \frac{T^t_{cpu}}{T^t_{epoch}} * (1 - \frac{f_t}{f_{max}}), \quad (6)$$

where IC_{max} is the maximum instruction count when cores work at the highest V/F level, f_{max} is the highest working frequency, and T^t_{cpu} is the time for executing instructions.

TABLE I
STATE SPACE OF DQN-LEARNER.

State Space		
BTR	Busy time ratio	$BTR_t \leftarrow BTR_t - 0.5$
$IPBC$	Instruction per busy cycle	$IPBC_t \leftarrow (IPBC_t - 0.5) * 2$
CPL	Cumulative performance loss	$CPL_t \leftarrow (CPL_t - TPL) * 10$
LVF	V/F setting in the last epoch	$LastVF_t \leftarrow LastVFID_t - 2$

PL_t only denotes a temporary performance decline in the epoch t. Thus, to achieve a pre-defined performance requirement at the end of the workload, cumulative performance loss (CPL) is defined to reflect the achieved performance degradation. CPL is calculated as

$$CPL_t = \frac{CPL_{t-1} * (T_{cum} - T^t_{epoch}) + PL_t * T^t_{epoch}}{T_{cum}}, \quad (7)$$

where T_{cum} is the cumulative time counting from the beginning of the workload.

Four state parameters are normalized before being input to the networks. The normalized transformation is shown in the last column in Table I. To simplify the computation, V/F levels in A are assigned integer action IDs $\{0, 1, 2, 3, 4\}$.

The reward r_t is defined as a combination of energy reward r^E_t and performance reward r^P_t, which is calculated as

$$r_t = \begin{cases} r^E_t & \text{for } CPL_t \leq TPL, \\ r^E_t - PF * r^P_t & otherwise. \end{cases} \quad (8)$$

where PF is the penalty factor and TPL is the pre-defined performance loss to limit the performance degradation.

Instead of simply using power or energy to represent the energy reward, we define another variable, that is, energy per instruction EPI. The function is calculated as

$$r^E_t = \frac{EPI_{f_{max}} - EPI_{f_t}}{EPI_{f_{max}}} = 1 - \frac{P_{f_t}}{P_{f_{max}} * (1 - PL_t)}, \quad (9)$$

where P_{f_t} is the core's power when the working frequency equals f_t. The performance reward r^P_t is equal to PL_t. When the performance requirement is not satisfied, that is $CPL > TPL$, the total reward is assigned a large negative value due to the product of r^P_t and PF, guiding the learning agent away from the unexpected policy in the near future. In this way, the pre-defined performance can be achieved and unsatisfying circumstances cannot last for a long time.

We implement our networks by 4 input neurons, one hidden layer of 10 hidden neurons, and 5 output neurons. The activation function of the hidden layer is leakyReLU.

Training epoch t for controlling is evoked periodically. Periodical training (PT) with comprehensive state observation enhance the stability and robustness of our online controller who knows nothing about applications and environmental changes. When the current policy does not fit the situation well due to inappropriate exploration or disturbance from the environment, the learning agent can adjust the model in one epoch's latency at the earliest after receiving a large negative reward. Additionally, PT allows control granularity to be tunable. There is a trade-off between control efficiency and training overhead including computation overhead and memory occupation. Fine granularity contributes to a high EDP reduction but leads to unexpected overhead. The latency of the hardware control also limits the choice of our period length. To consider the freshness of experiences, we adopt a combined experience replay (CER) algorithm in [14]. Whenever we sample a batch size of transitions, the lasted transitions of all the cores are added into the batch for training and the remaining experiences are sampled with priority following the mechanism in [15].

III. MULTI-DEVICE FEDERATED DQN-BASED POWER MANAGEMENT

A. Overview

Federated DQN-based (FDQN) power management among multiple devices is proposed to further improve the effectiveness of our control model, as shown in Fig. 2. Each device holds a learning agent to control the V/F levels of its own cores. The agent can make decisions and update parameters independently, as Section II-C illustrates. In this learning model, we divide the replay buffer into two parts. One is named the local replay buffer (LRB) that stores experiences from the local processor, and the other is called the remote replay buffer (RRB) receiving shared experiences from other devices. To avoid increasing the memory capacity and protect individual characteristics from being overlapped by others, we spare a small part from the original buffer to act as the RRB.

The key idea of our proposed method is to take full advantage of knowledge from other devices by sharing experiences. The DQN learning approach can be regarded as an infinite Q-table whose states are represented in a continuous format. As a result, exploration in DQN is much harder to be satisfied. Additionally, lack of experiences in the early stage may severely limit the improvement of our online learner and then slow down the converging speed. To tackle these problems, sharing knowledge, such as experiences, among devices can be an effective way.

B. Selective knowledge for sharing

We assume the initial states of all the devices are identical. After a period of learning-based control, different devices derive diverse policies due to the dynamic environment and random exploration. For a single device, we desire to receive the experiences that the local learner has not undergone or the well-learned experiences from an optimal policy derived by other devices. Through knowledge sharing, the device can achieve a more comprehensive exploration and thus derive a better converged result.

In our FDQN approach, we select the latest generated experiences to be shared. These experiences have high freshness, which means the distance between the generating policy and the current policy is small. Since the policy improves gradually with the training time, adopting the experiences of low freshness may degrade the learning effectiveness. Moreover, to reduce the communication overhead, we limit the number of sharing experiences to the number of cores.

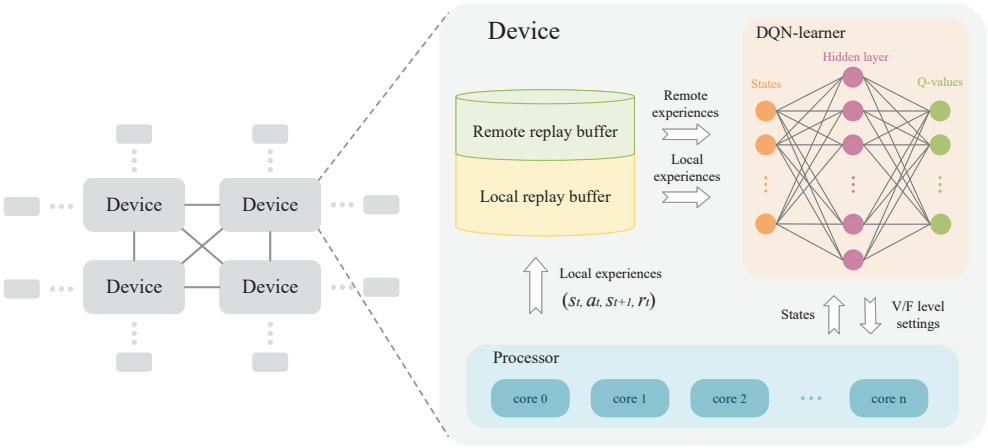


Fig. 2. Overview of federated DQN-based DVFS control.

Algorithm 1 Sharing Knowledge Updating Algorithm

Input: last state s_{t-1}, last action a_{t-1}, sharing buffer, replay buffer.
Output: a_t, sharing buffer, replay buffer.

while at learning epoch t **do**
  $s_t = compute\_state()$
  $r_{t-1} = compute\_reward()$
  sharing buffer.insert($\{s_{t-1}, a_{t-1}, r_{t-1}, s_t\}$)
  replay buffer.insert($\{s_{t-1}, a_{t-1}, r_{t-1}, s_t\}$)
  if sharing **then**
    $communication()$
  end if
  $update\_local\_policy()$
  if exploration **then**
    $a_t = random()\ \%\ action\ number$
  else
    $a_t = arg\max_a Q(s_t, a; \theta)$
  end if
end while

C. Knowledge sharing and policy updating

1) Sharing knowledge selecting and updating:

Algorithm 1 illustrates the procedure where the communication is combined with the local learning. At learning epoch t, current state s_t and last reward r_{t-1} induced by last state-action pair (s_{t-1}, a_{t-1}) are calculated according to the Eq. (4) to (9). Then, the completed transitions are inserted into the two buffers. Before updating the local policy, the device checks sharing requests. If sharing is enabled, the device will start communication, uploading and accepting the knowledge. Finally, the learner chooses actions by exploration or exploitation following ϵ-greedy algorithm. The exploration rate ϵ gradually decays with the progress of learning.

2) Remote knowledge accepting:

The received knowledge is inserted into the remote replay buffer (RRB) whose replacement mechanism is the same as a first-in-first-out buffer. Since we only assign a small size to the RRB, its replacement frequency is rather high, especially when the number of cooperative devices is large. In this way, out-of-date knowledge from others can be flushed in time, and thus the effectiveness of collaborative learning can be guaranteed.

Algorithm 2 Local Policy Updating Algorithm

Input: local replay buffer LRB, remote replay buffer RRB, network parameter θ.
Output: network parameter θ.

for $s = 1$ to batch size **do**
  $\{s_i, a_i, r_i, s_{i+1}\} = \begin{cases} sampling(LRB) & with\ probability\ \beta, \\ sampling(RRB) & otherwise. \end{cases}$
  $L_i(\theta) = (r_i + \gamma \max_{a'} Q(s_{i+1}, a'; \theta) - Q(s_i, a_i; \theta))^2$
end for
$\theta \leftarrow \theta + \alpha \cdot \frac{1}{batch\ size} \cdot \sum_i \bigtriangledown_\theta L_i(\theta)$

3) Local policy updating:

Algorithm 2 shows the mechanism of the local policy updating. We follow stochastic gradient descent to batch update the network's parameters. With the probability β, transitions are sampled from the local buffer, and with the probability $1-\beta$ from the remote buffer. The $sampling()$ function is an optimized experience replay method [14]. The remote buffer size equals the local replay buffer size multiplied by $\frac{\beta}{1-\beta}$, where β is set to less than 0.5.

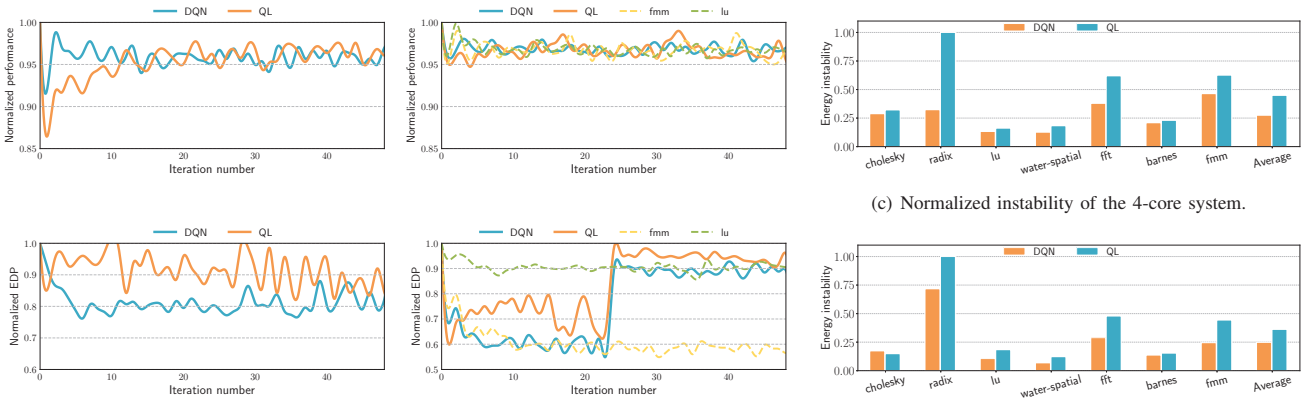
The values of the parameters for policy updating and action selection are listed as follows:

- learning rate: $\alpha = 0.5$;
- discount factor: $\gamma = 0.5$;
- penalty factor in reward: $PF = 20$;
- selection ratio: $\beta = 0.2$;
- initial exploration rate: $\epsilon_0 = 1.0$;
- batch size: $batch\ size = 20$;
- total replay buffer size: $RB\ size = 5 \times 10^3$;
- control period: $T_{epoch} = 1ms$.

IV. Experimental Results and Analysis

A. Experimental Setup

We evaluate our proposed methods in JADE simulator [16]. JADE integrates the power model based on McPAT [17]. We



(a) Radix.

(b) Lu-Water-spatial.

(c) Normalized instability of the 4-core system.

(d) Normalized instability of the 8-core system.

Fig. 3. Instability analysis of DQN and QL methods. (a-b) Normalized performance and EDP curve in 4-core systems. (c-d) Normalized instability of 4-core and 8-core systems.

assume that each processor has ARM-v8 cores with three levels of caches. The L1 and L2 caches are private and the L3 cache is shared. The L1 instruction and data caches are both 64 KB and 4-way associated, and the L2 cache is 256 KB and 8-way associated. The L3 cache is 16-way associated with a 2 MB capacity. The main memory is 8 GB. There are five V/F levels available in the system: 0.86V/2.8GHz, 0.78V/2.4GHz, 0.7V/2GHz, 0.64V/1.4GHz, 0.54V/1.2GHz. The assumed V/F levels follow the linear relationship in [18]. We use realistic applications from the COSMIC benchmark suit [19].

B. Efficiency of Deep Q-networks

In this subsection, we conduct experiments to show the improvement on instability and EDP reduction when the DQN method is adopted. We compare our DQN method with Q-learning-based (QL) and heuristic methods. The QL model is designed similarly to the DQN model. The difference between DQN and QL is that the state space of QL is partitioned into 140 discrete states using features in Table I and policy updating follows Eq. (1). The heuristic method is based on value prediction. We predict that the core's CPU time (T_{cpu}^{t+1}) in the next epoch equals the CPU time (T_{cpu}^{t}) in the current epoch, and the target PL (TPL) is expected to be reached in the next epoch. Then, according to Eq. (6) and (7), we compute the frequency (f_{t+1}) and select the next action a_{t+1}.

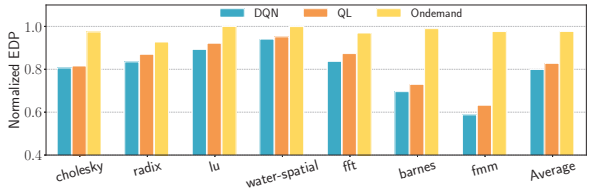
Figs. 3 and 4 show the compared results in 4-core and 8-core systems with target performance loss set to 5%. Applications are executed several times continuously and one execution is named as one iteration. Since we adopt periodical training, there are dozens or hundreds of epochs in one iteration according to the control period. The EDP and performance are normalized by the maximum EDP and performance without DVFS control.

We list two examples in Figs. 3(a) and 3(b) to show the learning process. In starting iterations of Fig. 3(a), both DQN and QL tend to explore new actions to cumulate knowledge. Thus, the performance curves may fluctuate below the required 95% target. When negative rewards are sensed, the learners applying DQN method can recover from the unexpected situation immediately due to the continuity of the state space and the high complexity of the policy function. By contrast, the QL-learner fails to pull the curves back to the required target since the Q-values in QL are updated independently and their initial values are usually set to 0. As a result, more learning epochs are required in QL before the learner recovers from the failed exploration and the asymptotic stability of QL is lower than DQN. In Fig. 3(a), the undesired EDP fluctuation of QL is over 10%. We further justify the robustness of our DQN method when significant changes happen. In Fig. 3(b), the green and yellow lines denote learning a single application without exchanging based on the DQN method. When the application changes to water-spatial from lu, DQN can recover from the significant change with a lower impulse in a shorter time compared with QL. Moreover, the normalized instability defined in Eq. (3) is shown in Figs. 3(c) and 3(d). Here, T is chosen as iteration 41 to 50 when the policy is optimized. DQN achieves around 14.5% stability improvement over QL, where the learning instability reduction of radix is up to 68%.

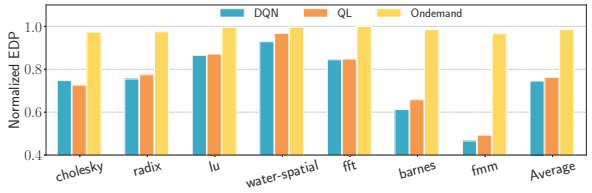
Apart from the instability improvement, the DQN-based method can achieve a larger EDP reduction as shown in Fig. 4. On average, around a 22.8% EDP reduction can be achieved when the DQN is adopted, which is 2.3% and 20.8% higher than the QL-based and the heuristic method, respectively.

C. Efficiency of Knowledge Sharing

To show the efficiency of knowledge sharing in FDQN, we conduct experiments on different numbers of communicating devices. In Fig. 5, QL and DQN denote applying QL-based and DQN-based methods on a single device without knowledge sharing. FDQN5, FDQN10, and FDQN15 denote applying FDQN-based methods with knowledge sharing among 5 devices, 10 devices, and 15 devices, respectively. Since experience exploration is crucial to the DQN-learner, with more devices sharing experiences, a lower EDP can be achieved. On average,



(a) 4-core systems.



(b) 8-core systems.

Fig. 4. Normalized EDP of the 4-core and 8-core systems.

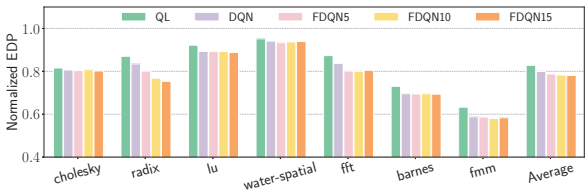


Fig. 5. Normalized EDP under varying communication devices number and learning methods.

the EDP reduction in FDQN15 is 4.8% higher than that of QL and 2% higher than of DQN.

We can find a decreasing trend of EDP when more devices participate in the federated learning group. To maintain local characteristics, we limit the size of the remote replay buffer. As a result, the improvement with an increase in device number is not significant in some applications.

D. Overhead Analysis

In this subsection, we analyze the overhead of our proposed method from two aspects including power and performance impact. Our DQN-based control is executed by one core on each device periodically. The energy consumption for computing DQN is around 0.16% to 0.35% of the total energy consumption for finishing the task including the application execution and DQN-based control, while the execution time only occupies around 0.58% to 0.98% of the total execution time. As for the communication overhead in knowledge sharing, the communication throughput of LTE [20] is around 6 and 13 Mb/s for uploading and downloading, respectively. Around 600 bits for one transferring consumes 240 μJ, which is less than 1% of the total energy consumption.

V. Conclusion

In this paper, we proposed a DQN-based DVFS control approach to reduce the EDP of the system while satisfying the user-specified performance requirement. On each device, the local DQN-learner periodically takes continuous state features as input and trains the model with combined experience replay. Compared to conventional Q-learning-based and heuristic methods, our proposed approach can enhance the stability and robustness of the learning process and achieve a higher EDP reduction under the same performance requirements. Through sharing knowledge among multiple devices, we can further improve the effectiveness of our learning-based method.

Acknowledgement

This work is partially supported by GRF16211418 and InnoHK ACCESS.

References

[1] A. Carroll *et al.*, "An analysis of power consumption in a smartphone." in *USENIX annual technical conference*, 2010.
[2] R. Ge *et al.*, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *IEEE Transactions on Parallel and Distributed Systems*, 2009.
[3] T. Kolpe *et al.*, "Enabling improved power management in multicore processors through clustered DVFS," in *2011 Design, Automation & Test in Europe*, 2011.
[4] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proceedings of the Linux Symposium*, 2006.
[5] H. Shen *et al.*, "Achieving autonomous power management using reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2013.
[6] R. A. Shafik *et al.*, "Learning transfer-based adaptive energy minimization in embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015.
[7] Z. Tian *et al.*, "Collaborative power management through knowledge sharing among multiple devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.
[8] F. M. M. ul Islam *et al.*, "Task aware hybrid DVFS for multi-core real-time systems using machine learning," *Information Sciences*, 2018.
[9] 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*, 2014.
[10] H. Hasselt, "Double Q-learning," *Advances in neural information processing systems*, 2010.
[11] L. Buşoniu *et al.*, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annual Reviews in Control*, 2018.
[12] F. Berkenkamp *et al.*, "Safe model-based reinforcement learning with stability guarantees," *arXiv preprint arXiv:1705.08551*, 2017.
[13] A. Saha *et al.*, "The interplay between stability and regret in online learning," *arXiv preprint arXiv:1211.6158*, 2012.
[14] S. Zhang and R. S. Sutton, "A deeper look at experience replay," *arXiv preprint arXiv:1712.01275*, 2017.
[15] T. Schaul *et al.*, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.
[16] R. K. V. Maeda *et al.*, "JADE: A heterogeneous multiprocessor system simulation platform using recorded and statistical application models," in *AISTECS*, 2016.
[17] S. Li *et al.*, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009.
[18] J. Henkel *et al.*, "New trends in dark silicon," in *Proc. DAC*, 2015.
[19] Z. Wang *et al.*, "A case study on the communication and computation behaviors of real applications in NoC-Based MPSoCs," in *ISVLSI*, 2014.
[20] J. Huang *et al.*, "A close examination of performance and power characteristics of 4g lte networks," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, 2012.