Smart Knowledge Transfer-based Runtime Power Management

Lin Chen[1], Xiao Li[1], Fan Jiang[1], Chengeng 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(GZ)
[†]*Corresponding author: jiang.xu@ust.hk*

***Abstract*—As Moore's law slows down, computing systems must pivot towards higher energy efficiency to continue scaling performance. Reinforcement learning (RL) performs more adaptively than conventional methods in runtime power management under varied hardware configurations and varying software workloads. However, prior works on either model-free or model-based RL approaches face a non-negligible challenge: relearning the policies to adapt to the new environment is unacceptably time-consuming, especially when encountering significant variances in workloads or hardware configurations. Moreover, existing research on accelerating learning has focused on the speedup while largely ignoring the efficiency degradation of the results. In this paper, we present a smart transfer-enabled Q-learning (STQL) approach to boost the learning process and guarantee the learning efficiency through a contradiction checking mechanism, which wisely evicts inappropriate transferred knowledge. Experiments on realistic applications show that the proposed method can speed up the learning process to up to 2.3x and achieve a 6.2% energy-delay product (EDP) reduction compared to the state-of-the-art design.**

***Index Terms*—power management, reinforcement learning, online learning, transfer learning, multicore system**

I. Introduction

Power wall has become one of the primary challenges that modern computing systems are hitting. This is especially true on mobile devices, where the problem is further aggravated by the constrained power supply. In response, the system-wide power management has gained much attention from both the industry and academia for harvesting the potential improvement in energy efficiency. Additionally, cores are reported to be the primary energy consumers compared to all other components in the system according to the power analysis in [1]; thus, improving the energy efficiency of the cores is critical to defeating the power wall challenge.

Dynamic voltage and frequency scaling (DVFS) is one of the most widely used techniques for improving the energy efficiency of cores. It allows software or hardware to dynamically monitor the runtime status of cores and control the Voltage/Frequency (V/F) levels. Existing heuristic methods [2], [3] based on power profiling and short-term prediction are observed to be inefficient in adapting to stochastic variances from environments and workloads, leading to considerable energy efficiency optimization space. To fill the inadaptability gap, reinforcement learning (RL)-based approaches with self-calibration mechanisms [4]–[10] have been proposed and show significant energy efficiency improvement over heuristic methods. However, the notoriously long training time of RL has greatly limited its application. An optimal policy can only be achieved after an extensive exploration of the whole state-action space, which is highly time-consuming [11] and results in a large computational overhead. Worse yet, the ever-increasing complexity of both hardware and software exacerbates the implementation of sufficient exploration.

In our runtime power management for multicore processors, efficient learning is greatly challenging and time-consuming since the state-action space of RL grows exponentially as the number of cores increases. Moreover, the learning time is further extended by the dynamic variances of workloads. During the online management process, both intra- and inter-workload variances induce burdensome exploration to relearn and evict the unsuitable decisions, causing the system to be trapped in a suboptimal or even wrong policy for a lasting time. In the worst case, the slow learning speed may eliminate the overall energy efficiency profits brought by the sophisticated RL, which essentially restricts the commercial adoption of RL-based power management. Unfortunately, most previous works ignore the inherent dilemma [4]–[8].

To accelerate the learning process with workload variances, we introduce transfer learning (TL) to reuse knowledge from past and similar tasks. Knowledge transfer in [11]–[13] is represented as a special exploration strategy by adopting knowledge from previous experiences. The learning agent in [14] transfers the knowledge in similar states between the source task and the target task, and the agent in [10] transfers from its neighbors. Despite the learning acceleration, knowledge from previous tasks can generate policy contradiction and suboptimality at a high probability. Reusing inappropriate information can lead to suboptimal or even failed decisions, which severely degrades the learning efficiency. However, the existing TL-based works [10]–[14] only focus on similarity function construction and prior knowledge selection for acceleration while ignoring the appropriateness of the transferred knowledge, which is as important as acceleration.

In this paper, we propose a smart transfer-enabled Q-learning-based (STQL) approach to conduct runtime power management for multicore systems. A Q-learning (QL) agent periodically monitors multiple runtime information about cores and updates its policy. When a significant change happens, transfer learning (TL) is enabled to help the QL agent adapt to the new environment at a fast speed. Additionally, in order to eliminate the learning suboptimality induced by past knowledge, a novel and automatic contradiction checking (CC) mechanism is proposed in our STQL. We compare our STQL with the state-of-the-art TL-based methods, and the results show that our STQL can not only speed up the learning process but also guarantee learning efficiency similar to learning from scratch. The main contributions of our work are as follows:

- We propose a smart transfer-enabled Q-learning-based (STQL) approach to reduce the energy-delay product

978-3-9819263-7-8/DATE23/© 2023 EDAA

(EDP) of multicore systems with the ability to tackle significant changes during the online process.
- We propose a novel contradicting checking (CC) mechanism to smartly reuse the knowledge from past tasks and minimize the impact of suboptimality.
- We comprehensively analyze the intra- and inter-workload variances and illustrate the necessity of our smart transfer learning.
- We quantitatively evaluate the effectiveness of our proposed methods compared with state-of-the-art methods.

II. Reinforcement Learning-based Power Management

A. Reinforcement Learning Basics

The dynamic and adaptive advantages of reinforcement learning (RL) make this machine learning method widely used in automatic industry control, game decision-making, trading and finance, etc. We use Q-learning, one of the most popular RL algorithms, to conduct DVFS control. The QL agent keeps a Q-value for every state-action pair to evaluate the pair's performance according to the current and the future rewards. At each learning epoch t, the Q-value is 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 related reward r_t. α denotes learning rate, and γ denotes the discount factor that represents how much the future reward counts.

B. Exploration Strategy

A well-known ϵ-greedy exploration rule is adopted in our proposed QL-based power management. The exploration and exploitation rotation mechanism is shown as follows:

$$\pi(s) = \begin{cases} random\ action\ in\ A & p < \epsilon(s)\ (exploration), \\ \underset{a\in A}{argmax} Q(s,a) & otherwise\ (exploitation), \end{cases} \quad (2)$$

where p is a random variable ranging in $[0,1]$. When $p<\epsilon(s)$, we randomly choose one action from action space A; otherwise, we select the action with the maximum Q-value.

In most existing works, the exploration probability is degraded by visited times (VT) to promote the convergence process. However, we find VT can lead to inadequate exploration due to some extremely visited states. As a result, we first introduce explored degree (ED) to guarantee the exploration quality. The exploration threshold $\epsilon(s)$ in Eq. (2) is defined as

$$\epsilon(s) = 1/(ED(s)+1)^{\theta}, \quad (3)$$

where $ED(s)$ denotes the explored degree of state s ranging in $[0,|A|-1]$, $|A|$ equals the total number of actions, and θ is a degradation factor. For each state s, when action a is selected and the corresponding $Q(s,a)$ is updated, $ED(s)$ is incremented by 1. $ED(s)$ reflects the expertness level of the current agent to the state s, where large $ED(s)$ value means high expertness and confidence level.

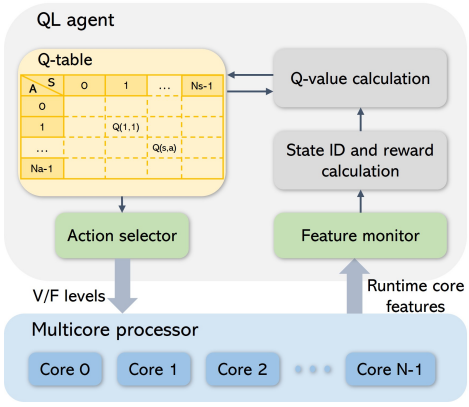


Fig. 1. Overview of the QL-based power management.

TABLE I
State Space of the QL agent.

State Space		
CUR	CPU Occupation Rate	$CUR_t = T^t_{CPU}/T_{epoch}$
CPL	Cumulative performance loss	$CPL_t = \frac{CPL_{t-1}*(T_{cum}-T_{epoch})+PL_t*T_{epoch}}{T_{cum}}$
LVF	V/F setting in the last epoch	$LVF \in \|A\|$

C. QL-based Power Management Formulation

1) Overview and Objective: We first illustrate how to formulate the basic part of our power management for a multicore processor, as shown in Fig. 1. Per-core DVFS is achieved by one shared QL agent to leverage the working divergence among cores. At each learning epoch t, a feature monitor observes the runtime features of cores to compute state s_t and reward r_{t-1}. After Q-values are updated, the agent selects action a_t following ϵ-greedy strategy. The learned policy is represented in a tabular format (Q-table), where Q-values for each state-action pair are stored.

The objective of our QL agent is to reduce the energy-delay product (EDP) of the multicore processor while satisfying user-defined performance requirements, e.g., 5% or 10% limitation of performance loss. After the primary objective is achieved, we further introduce transfer learning to deal with the significant variances of workloads during the online control process, which will be discussed later in Section III.

2) State Space: In this paper, we extract three features of cores to construct the state space S. Definitions and calculation functions are shown in Table I. CUR uses CPU time to reflect the core utilization degree in epoch t. CPL_t represents the current performance loss in a cumulative way, where T_{cum} is the execution time accumulated since the start of learning. PL_t denotes the discounted instruction counts (ΔIC) when cores work at a low V/F level with respect to the highest V/F level.

We monitor multiple state features periodically to allow the learning agent to obtain comprehensive information about the cores, which improves the efficiency and robustness of the QL-based method. After three features are calculated, an integer state ID is assigned through a discrete partition.

3) Reward Function: We divide our objective into two parts to formulate the reward function: energy minimization r_t^{enrg} and performance target satisfaction r_t^{perf}.

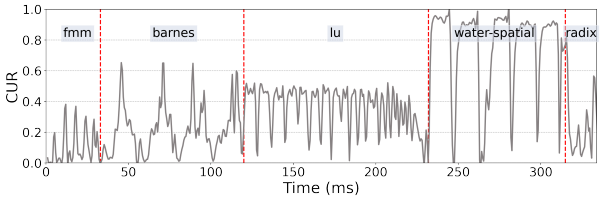


Fig. 2. CPU occupation ratio (CUR) variances.

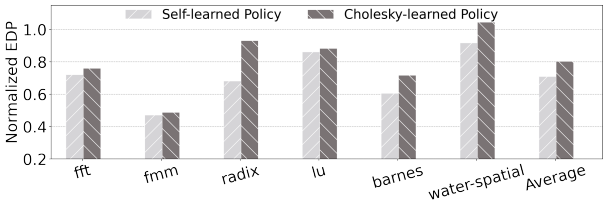


Fig. 3. Efficiency of self-learned policy and cholesky-learned policy.

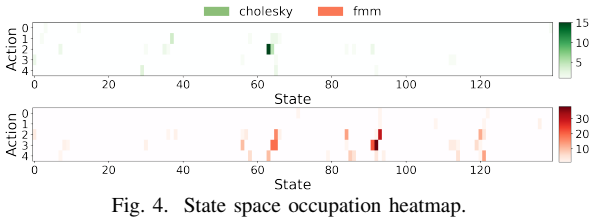


Fig. 4. State space occupation heatmap.

Energy reward r_t^{enrg} represents the power saving under the current V/F level with respect to the highest V/F level at epoch t, which is define as

$$r_t^{enrg} = 1 - \frac{P_{f_t}}{P_{f_{max}}(1 - PL_t)}, \tag{4}$$

where P_{f_t} is the power of the core working at frequency f_t. Since the low frequency causes the performance loss at epoch t, we multiples $P_{f_{max}}$ with $(1 - PL_t)$ to keep identical executed IC when computing the power loss under different V/F levels.

Performance reward r_t^{perf} uses CPL_t to evaluate the performance satisfaction degree at epoch t, which is defined as

$$r_t^{perf} = \begin{cases} -PF * PL_t & \text{for } CPL_t > TPL, \\ 0 & otherwise. \end{cases} \tag{5}$$

When CPL_t is larger than the target performance loss TPL indicating the required performance is not satisfied, we assign r_t^{perf} a large negative value by multiplying PL_t with a penalty factor PF, guiding the learning agent away from the undesired cases in the near future. Otherwise, r_t^{perf} equals 0. As a result, reward r_t is calculated as

$$r_t = r_t^{enrg} + r_t^{perf}. \tag{6}$$

III. Smart Transfer Learning-based Reinforcement Learning for Runtime Power Management

A. Motivation of Transfer Learning

During our online power control process, the learning agent will encounter different levels of workload variances. Fig. 2 shows the CPU occupation ratio (CUR) variances of 5 applications under periodical monitoring windows. In an overall view, the execution patterns vary obviously among different applications, which is reflected in divergent CUR levels and changing frequencies. For intra-application variances, most of them are light-weighted and regular, which can be handled by QL agents with automatic state monitoring and policy revising. Whereas for inter-application variances, execution patterns such as feature levels diverge significantly when application switching happens. If we directly reuse the policy learned by previous applications neglecting the application variances, the learning efficiency is degraded by 9.4% averagely compared to the self-learned policy, as shown in Fig. 3. Here, the self-learned policy denotes learning the policy from scratch without adopting previous experiences. The suboptimality induced by inter-application variances calls for relearning process to evict the inappropriate knowledge, which is highly time-consuming. Therefore, most existing works introduce transfer learning to accelerate or remove the relearning process. However, the suboptimality is actually not eliminated and is even compromised for acceleration. Considering the unsolved gap, we propose a variance-aware and efficiency-guaranteed transfer learning-based approach for runtime power management.

To quantitatively analyze the factors that influence the success of transfer learning, we evaluate the inter-application discrepancy using the degree of overlap in the state space. We display the state space occupation map in Fig. 4, where the color values denote the visited times of corresponding state-action pairs. We observe the two applications, cholesky and fmm, occupy distinct state locations in most instances with a small quantity of overlapped cases. When application switching happens, the existence of the non-overlapping state space means the learning agent visits new states that are not experienced while executing the previous applications. This discrepancy requires new exploration, and ignoring the issue is bound to cause suboptimal learned results. As for the overlapping state space, there exist contradictory and suboptimal cases due to different execution and transition patterns of the applications. In more detail, for an identical state s, the policy learned from the previous application selects a_p as the best action, while the new application expects a_n to achieve larger rewards. We define a case as contradictory when the difference between a_p and a_n under the same state s exceeds $|A|/2$; otherwise, the case is recorded as a suboptimal case.

We count the ratio of the three cases between cholesky and other applications. There are 49.52% overlapping states on average. Among these overlapping states, the suboptimal ratio and contradictory ratio are 58.98% and 14.48%, respectively, which explains the suboptimal results shown in Fig. 3. In summary, around 50% overlapping ratio indicates a substantial reusing space for knowledge transfer to accelerate the learning process. Nevertheless, around 15% contradictory ratio emphasizes the necessity of wise transfer to guarantee the learning efficiency. Therefore, we propose a novel and smart transfer approach with satisfying abilities to deal with the above two issues.

B. Overview

Fig. 5 shows the overview of our proposed STQL-based power management. The QL agent illustrated in Section II-C is reserved as the basic QL part. Moreover, an expert policy is

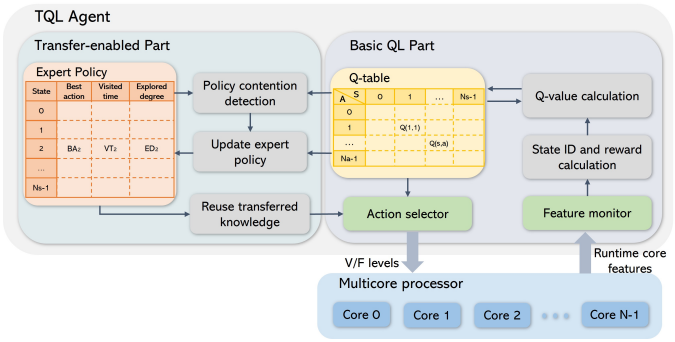


Fig. 5. Overview of the STQL-based power management.

introduced to enable knowledge transfer. In the expert policy, best action BA, visited time VT, and explored degree ED are recorded for each state, where BA denotes the action with maximum Q-value, VT is a counter recording the accumulated time that the learning agent visits the state, and ED reflects the exploration expertness. The policy contention detection mechanism distinguishes the contradiction between the current Q-table and the expert policy and updates the expert policy when a contradictory case happens. The action selector makes decisions based on the Q-table and transferred knowledge.

In our approach, past knowledge is formulated as an expert policy. Compared with the state-of-the-art transfer-based methods, where a complete Q-table is reserved for transfer, our proposed method largely saves memory overhead in a concise representation way along with extra guidance (VT and ED) for wise reuse.

C. Policy Updating and Contradicting Checking

The local and expert policies updating mechanism is illustrated in Algorithm 1. At a learning epoch t, the basic QL part computes the current state ID s_t and the reward r_{t-1}. Then the Q-value $Q(s_{t-1}, a_{t-1})$ is updated according to Eq. (1) (Line 2-4). If the learning agent is informed that an application exchange happens, the expert policy extracts knowledge from the latest Q-table and resets the Q-table afterward (Line 5-7). Otherwise, we check the contradiction between the local and expert policies to decide whether to evict the outdated knowledge of the expert policy (Line 9-12). Two contradiction-checking conditions are defined: 1) the best action of the expert policy differs from that of the local policy (expert policy[s_{t-1}].BA $!= a_{best}$); 2) the explored degree of the expert policy is less than or equals that of the local policy (expert policy[s_{t-1}].ED $<= ED(s_{t-1})$). Either of the two conditions will trigger an update of the expert policy.

D. Transfer-based Action Selecting

We follow the strategy illustrated in Section II-B to decide whether to explore or exploit. If the transfer enabled flag is true, we can exploit both the expert policy and the local policy to select actions. Since the direct reuse of the previous policy can lead to severe suboptimality, as shown in Fig. 3, we define two boundaries to limit the adoption ratio of the expert policy: 1) expert policy[s_t].ED $> ED(s_t)$; 2) $VT(s_t) < |A|$. In this way,

Algorithm 1 Policy Updating Algorithm

Input: last state s_{t-1}, last action a_{t-1}, current state s_t, reward r_{t-1}, Q-table, expert policy.
Output: Q-table, expert policy.

1: **while** at learning epoch t **do**
2: $s_t = compute\_stateID()$
3: $r_{t-1} = compute\_reward()$
4: $Q(s_{t-1}, a_{t-1}) = (1-\alpha)Q(s_{t-1}, a_{t-1}) + \alpha[r_{t-1} + \gamma \max_{a'} Q(s_t, a')]$
5: **if** transfer enabled **then**
6: expert policy = $extract\_policy()$
7: $reset\_Qtable()$
8: **else**
9: $a_{best} = argmax_a Q(s_{t-1})$
10: $ED(s_{t-1}) = get\_explored\_degree(s_{t-1})$
11: **if** expert policy[s_{t-1}].BA $!= a_{best}$ or expert policy[s_{t-1}].ED $<= ED(s_{t-1})$ **then**
12: $update\_expert\_policy(s_{t-1})$
13: **end if**
14: **end if**
15: **end while**

we tend to reuse the expert policy that has a higher expertness level at the early time after application switching occurs. As the learning progresses, the local QL agent accumulates knowledge and checks the contradiction with the expert policy. Gradually, the local policy is forced to take the lead in choosing actions.

IV. Experimental Results and Analysis

A. Experimental Setup

We evaluate our proposed methods in JADE simulator [15] with power model based on McPAT [16]. We assume that each processor has multiple 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 [17]. The simulator uses realistic applications from the COSMIC benchmark suit [18].

B. Baseline Methods

1) Direct Transfer (DT): DT is a simple transfer method that uses the learned policy from the source task as the initial policy of the target task. In transfer-based QL, the Q-table learned by the previous application is directly reused as the initial policy when a new application comes.

2) Similar Transfer (ST): Similar transfer (ST) is a conservative and pervasive way to reuse knowledge from similar states in past tasks [12], [14], [19]. One of the primary challenges in ST is to design a similarity function $sim\_func()$ and transform the knowledge from the past tasks to the target task. After

$sim\_func()$ is constructed, we select actions according to the following mechanism:

$$a = \begin{cases} \prod_{past}(s) & sim\_func(s) \in S_{past}, \\ \epsilon - greedy(\prod_{new}(s)) & otherwise, \end{cases} \quad (7)$$

where S_{past} denotes the state space of past tasks. In our evaluation, we define $sim\_func()$ as $s = sim\_func(s)$, which is adopted in [14].

3) Intra-Task Learning Transfer (ITLT): ITLT is a state-of-the-art method for runtime power management using transfer learning [10]. In ITLT, transferred knowledge is gathered from adjacent states. When task changes happen, the learning agent follows three steps to conduct ITLT: 1) find explored states from neighbors; 2) compute weights according to the distance; 3) add the weighted Q-values of the explored states to the Q-value of the current state.

C. Learning parameters

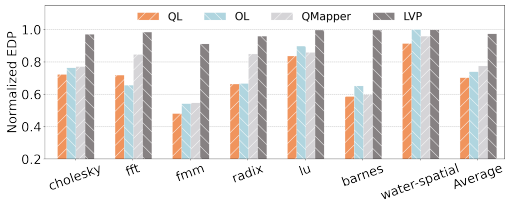
The values of learning parameters of our proposed STQL-based method are listed as follows:

- learning rate: $\alpha = 0.9$;
- discount factor: $\gamma = 0.1$;
- penalty factor in reward: $PF = 20$;
- initial exploration rate: $\epsilon_0 = 1.0$;
- state feature partition factors:
  - CUR partition: 0.2, 0.5, 0.8
  - CPL partition: -0.02, -0.01, -0.005, 0.005, 0.01, 0.02
- state count: $|S| = 140$
- action count: $|A| = 5$
- control period: $T_{epoch} = 1ms$
- target performance loss: $TPL = 10\%$

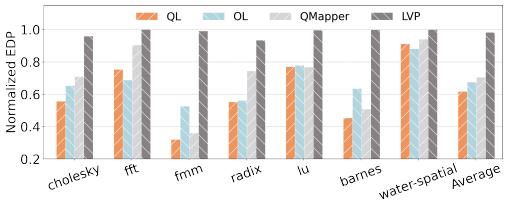
D. Efficiency of QL-based Power Management

We compare our proposed QL-based method with a prediction-based method (LVP), a previous QL-based method (QMapper) [20], and an offline-based method (OL) to show the advantage of our QL in runtime power management. Last value prediction (LVP) predicts 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. Based on the prediction, we compute the frequency to be selected at the next epoch (f_{t+1}). QMapper only uses CUR to construct the state space. OL trains a well-learned policy using abundant applications offline, which is usually treated as Oracle in imitation learning or reinforcement learning [21]. Here, to evaluate the efficiency of an offline-learned policy for an unknown application, we train the policy using our proposed QL-based method, excluding the unknown application.

As shown in Fig. 6, our QL can achieve around 29.7% and 38.3% EDP reduction on 4-core and 8-core systems, respectively, which is 7.3% and 8.8% higher than QMapper. The improved results indicate the advantage of combining multiple state features as state space. QL also outperforms OL by 3.7% and 5.8%, which further corroborates our motivations to adopt transfer learning (TL) for unpredictable variances. If Oracle itself cannot achieve optimal results, the methods supervised by Oracle with the objective to approximate Oracle are bounded to be suboptimal. LVP achieves minimal EDP improvement due to its worst capability to adapt to dynamic workload variances.



(a) 4-core



(b) 8-core

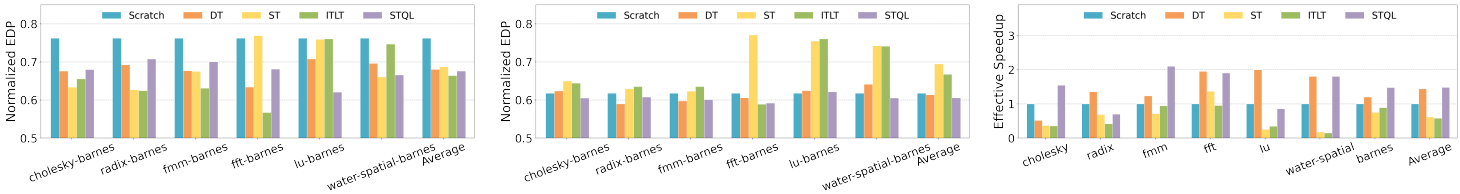
Fig. 6. Normalized EDP of the 4-core and 8-core systems.

E. Efficiency of STQL-based Power Management

In this subsection, we compare our proposed STQL-based method with the baseline methods: scratch, DT, ST, and ITLT. When an application change happens, the scratch-based method resets the learning parameters and initializes the policy.

1) Early stage of TL: We evaluate the EDP improvement at the early stage when transfer learning is enabled to present the acceleration profits brought by different TL-based methods. When application switching happens, TL is desired to reuse the accumulated knowledge from the past task and achieve an appreciable EDP reduction at a fast speed.

Fig. 7(a) shows the averaged EDP results of the first iteration. X-barnes denotes that application barnes is executed after application X is executed for several iterations. On average, all the transfer-based methods achieve higher EDP reduction at the early stage compared with scratch. DT accepts the largest portion of policy learned by the previous application. Hence, DT can reduce EDP at a fast speed when the overlapping state ratio is high, such as fft-barnes and fmm-barnes. However, due to the existence of suboptimal and contradictory cases, directly adopting the knowledge from these cases can restrict the reducing pace instead, such as lu-barnes. ST only transfers the policy of overlapping states between the two applications. Since new exploration only happens in non-overlapping states, the ST agent is trapped in wrong decisions which are made by the previous policy. Hence, ST suffers largely from contradictory cases, as shown in fft-barnes and lu-barnes. As for ITLT, transferred knowledge is from adjacent states, which is more aggressive than ST, resulting in a higher reuse ratio for acceleration. Nevertheless, the harmful effect brought by the contradictory case is exacerbated as well, such as lu-barnes and water-spatial-barnes. Unlike the speedup-intended methods, our proposed STQL reserves the property of ST for acceleration and boosts the ability to distinguish different kinds of state cases for efficiency guarantee. Although our STQL compromises



(a) Early stage when TL is enabled (Acceleration). (b) Later stage when TL is enabled (Learning efficiency). (c) Effective Speedup of TL-based methods.

Fig. 7. Efficiency of STQL-based power management.

acceleration profits for efficiency enhancement, it still achieves appreciable EDP improvement over scratch, 8.6% on average.

2) Later stage of TL: Despite the learning speed, the sub-optimality of the learned policy after transferring is also a significant concern. Therefore, we evaluate the EDP improvement in the later stage to display the learning efficiency of the measured TL-based methods, expecting the extra introduction of TL not to harm the learned results. Fig. 7(b) shows the average EDP from iteration 5 to 10. Since DT, ST, and ITLT cannot deal with the suboptimal and contradictory cases, their EDP reduction is worse than scratch in some applications, no matter how fast they reduce the EDP at the early stage. The contradiction-checking mechanism in our method takes effect after several learning epochs when the local policy has accumulated some knowledge. On average, STQL achieves 39.5% EDP reduction in the later stage, 9% and 6.2% higher than ST, ITLT, respectively.

3) Effective Speedup: To quantitatively measure the degree of acceleration, we calculate the iteration number when the EDP is reduced under a threshold. The converged threshold is defined as a standard EDP level achieved by scratch plus 1% of the standard level. Then, we define the effective speedup as the inverse of the converged iteration number, as shown in Fig. 7(c). Among various kinds of applications, STQL can achieve around 1.5x speedup on average, while in fmm, the speedup can reach up to 2.1x, which is 2.3x of ITLT. The speedup of ST and ITLT is less than 1, indicating the methods can not reach the scratch level and cause suboptimality in learned policies. Since our effective speedup is measured in iteration number, the saving time equals speedup multiplied by the execution time for one iteration, whose value is very considerable, especially for complex applications.

F. Overhead Analysis

In this subsection, we analyze the overhead of our proposed method. The power and performance impact of the basic QL is minimal since the overhead is mainly caused by calculating the state, reward, and Q-value. We execute our STQL-based DVFS control on one core. The energy consumption for updating Q-values and selecting actions is around 0.05% to 0.11% of the total energy consumption for finishing the task, including the application execution and STQL-based control, while the execution time only occupies around 0.19% to 0.32% of the total execution time. The major overhead is from memory occupation. Around 5.5KB of memory is required to store a Q-table with a size of 140x5. An expert policy in STQL introduces an extra 1.6KB memory overhead, where we only store int values in the expert policy to save the memory cost. Compared with ITLT, our method saves around 3.9KB cost.

V. Conclusion

In this paper, we propose a STQL-based approach for runtime power management. Transfer learning with contradiction checking is enabled when significant workload variances happen in the systems. Compared to state-of-the-art TL-based methods, our proposed approach can accelerate the learning process while guaranteeing learning efficiency at the same time.

Acknowledgement

This work is partially supported by HKUST(GZ) and ACCESS.

References

[1] H.-Y. Cheng, J. Zhan, J. Zhao, Y. Xie, J. Sampson, and M. J. Irwin, "Core vs. uncore: The heart of darkness," in *52nd DAC*. IEEE, 2015.
[2] T. Kolpe *et al.*, "Enabling improved power management in multicore processors through clustered DVFS," in *DATE*. IEEE, 2011.
[3] V. Pallipadi *et al.*, "The ondemand governor," in *Linux Symposium*, 2006.
[4] Y. Tan *et al.*, "Adaptive power management using reinforcement learning," in *IEEE/ACM ICCAD-Digest of Technical Papers*, 2009.
[5] H. Shen *et al.*, "Achieving autonomous power management using reinforcement learning," *ACM TODAES*, 2013.
[6] R. A. Shafik *et al.*, "Learning transfer-based adaptive energy minimization in embedded systems," *IEEE TCAD*, 2015.
[7] F. M. M. ul Islam *et al.*, "Task aware hybrid DVFS for multi-core real-time systems using machine learning," *Information Sciences*, 2018.
[8] R. Ye *et al.*, "Learning-based power management for multicore processors via idle period manipulation," *IEEE TCAD*, 2014.
[9] Z. Tian *et al.*, "Collaborative power management through knowledge sharing among multiple devices," *IEEE TCAD*, 2018.
[10] D. Jenkus *et al.*, "Runtime energy minimization of distributed many-core systems using transfer learning," in *DATE*. IEEE, 2022.
[11] L. A. Celiberto Jr *et al.*, "Using transfer learning to speed-up reinforcement learning: a cased-based approach," in *2010 Latin American Robotics Symposium and Intelligent Robotics Meeting*. IEEE, 2010.
[12] J. L. Carroll *et al.*, "Fixed vs. dynamic sub-transfer in reinforcement learning." in *ICMLA*, 2002.
[13] F. Fernández *et al.*, "Probabilistic policy reuse in a reinforcement learning agent," in *AAMAS*, 2006.
[14] M. E. Taylor *et al.*, "Behavior transfer for value-function-based reinforcement learning," in *AAMAS*, 2005.
[15] R. K. V. Maeda *et al.*, "JADE: A heterogeneous multiprocessor system simulation platform using recorded and statistical application models," in *AISTECS*, 2016.
[16] S. Li *et al.*, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009.
[17] J. Henkel *et al.*, "New trends in dark silicon," in *Proc. DAC*, 2015.
[18] Z. Wang *et al.*, "A case study on the communication and computation behaviors of real applications in NoC-Based MPSoCs," in *ISVLSI*, 2014.
[19] M. G. Madden *et al.*, "Transfer of experience between reinforcement learning environments with progressive difficulty," *Artificial Intelligence Review*, 2004.
[20] A. Monteiro *et al.*, "Qmapper: Dynamic power and performance management in virtualized web servers clusters," in *IEEE Eighth LADC*, 2018.
[21] U. Gupta *et al.*, "A deep q-learning approach for dynamic management of heterogeneous processors," *IEEE Computer Architecture Letters*, 2019.