Multi-Agent Reinforcement Learning for Thermally-Restricted Performance Optimization on Manycores

Heba Khdr, Mustafa Enes Batur, Kanran Zhou, Mohammed Bakr Sikal, Jörg Henkel
Karlsruhe Institute of Technology, Karlsruhe, Germany
{heba.khdr, bakr.sikal, henkel}@kit.edu, {mustafa.batur, kanran.zhou}@student.kit.edu

***Abstract*—The problem of performance maximization under a thermal constraint has been tackled by means of dynamic voltage and frequency scaling (DVFS) in many system-level optimization techniques. State-of-the-art ones have exploited Supervised Learning (SL) to develop models that predict power and performance characteristics of applications and temperature of the cores. Such predictions enable proactive and efficient optimization decisions that exploit performance potentials under a temperature constraint. SL-based models are built at design time based on training data generated considering specific environment settings, i.e., processor architecture, cooling system, ambient temperature, etc. Hence, these models cannot adapt at runtime to different environment settings. In contrast, Reinforcement Learning (RL) employs an agent that explores and learns the environment at runtime, and hence can adapt to its potential changes. Nonetheless, using an RL agent to perform optimization on manycores is challenging because of the inherent large state/action spaces that might hinder the agent's ability to converge. To get the advantages of RL while tackling this challenge, we employ for the first time multi-agent RL to perform thermally-restricted performance optimization for manycores through DVFS. We investigated two RL algorithms—Table-based Q-Learning (TQL) and Deep Q-Learning (DQL)—and demonstrated that the latter outperforms the former. Compared to the state of the art, our DQL delivers a significant performance improvement of 34.96% on average, while also guaranteeing thermally-safe operation on the manycore. Our evaluation reveals the runtime adaptability of our DQL to varying workloads and ambient temperatures.**

I. INTRODUCTION

Thermal restriction has become one of the main design constraints of processors due to the severe impacts of elevated temperatures on reliability [1]. Most processors incorporate a hardware unit called Thermal Control Circuitry (TCC) to enforce a thermal constraint [2]. When the temperature of any core on the processor violates the predefined thermal constraint, TCC sets the voltage/frequency (V/f) levels of all cores to the minimum level, regardless of which core is causing this violation. Obviously, this degrades the performance of all running applications on the chip. To bypass such conservative circuit-level countermeasures, many research studies propose to manage the on-chip temperature at system level, where relevant knowledge about the system is available and can be exploited to avoid unnecessary performance loss. In particular, state-of-the-art system-level optimization techniques employ models to estimate the performance of applications and the resulting power and temperature if they would run at different V/f levels. Such estimations help to proactively adjust the V/f levels of the cores accordingly, so that overall system performance is maximized under the thermal constraint.

Traditionally, power and performance models are built through extensive profiling of applications at design time [3]. Obviously, such modeling is feasible only for a priori known applications. For temperature models, the well-known RC thermal networks [4] are widely used. However, these models require an accurate floorplan of the chip and a precise description of the cooling system, information that is often not available for system-level techniques.

Recently, SL has been applied to predict power [5] and performance [6] even for unknown applications and to estimate temperature [7] without the need for information about the chip floorplan and the cooling system. Based on SL models, smart DVFS policies [8] are proposed, with the aim of maximizing the overall system performance under a thermal constraint. These models, however, are built at design time based on training data that is generated on specific environment settings, e.g., processor architecture, cooling system, and ambient temperature. Hence, they cannot adapt to different settings at runtime. To make these models generalize to unseen settings, training data needs to be collected considering many of these settings, i.e., running experiments on different processors, cooling systems, and various ambient temperatures. Nonetheless, this approach is impractical, which explains why it has not been adopted by the state-of-the-art techniques.

Contrarily, RL empowers online and adaptive learning by employing an agent that interacts with the environment at runtime and learns how to achieve the given objective by a trial-and-error approach. Specifically, an RL agent observes the state of the environment and takes an action accordingly. Then, the agent receives either a reward or a penalty, depending on the effectiveness of its action. Recent system-level optimization techniques have applied RL, but most of them have targeted power and energy optimization [9], [10]. For thermal optimization, few techniques have been proposed in the literature, but targeted only the problem of temperature minimization under performance constraints for multicores [11], [12]. None of the state-of-the-art techniques has employed RL to tackle the problem of performance maximization under a thermal constraint. *Our paper is the first to do that*. To this end, we investigated two popular RL methods—TQL and DQL—to examine their efficacy in achieving our targeted goal.

Challenges and Contributions: Using RL to achieve the targeted optimization for manycores is challenging. First, due to the large number of cores on a many-core processor, RL requires large state/action spaces, that is, the state space will include performance metrics and temperatures of all cores, while the action space will include all possible combinations of

979-8-3503-4859-0/DATE24/© 2024 EDAA

available V/f levels of all cores. Such large state/action spaces might hinder the convergence of the RL agent. To cope with this challenge, we apply multi-agent RL, where each agent is responsible for selecting the V/f level of one core. Hence, the action space of each agent contains only the available V/f levels on the core. For the state space, it is intuitive to involve measurements for the performance of the agent's core and its temperature. However, due to the heat transfer between the cores, the decision of one agent on one core might lead to thermal violations on the cores of other agents, especially the neighboring ones that surround this core. Therefore, we add the temperature of the neighboring cores to the state of each agent. Another challenge is the time required by RL agents to sufficiently learn the environment and be capable of taking effective actions. To accelerate the learning, we enable *knowledge sharing* among the agents [13]. In summary, our **novel contributions** are:

- We propose a RL-based thermally-restricted performance optimization technique for manycores.
- To provide scalability on manycores, we employ multi-agent RL, where each agent is responsible for setting the V/f level of one core. We enable knowledge sharing among the agents to accelerate their learning.
- We investigate two RL methods—TQL and DQL—to compare their efficacy in solving the targeted optimization.

II. Related Work

The advent of machine learning in recent years has enabled the development of smarter DVFS techniques, opening new avenues for power/energy, performance and thermal optimization. Within this scope, a large body of research has opted for the SL paradigm. The authors in [6] proposed a Deep Neural Network (DNN)-based DVFS technique for heterogeneous multi-core processors. Depending on the execution phases of the running applications, their trained DNN is invoked to predict the per-core V/f levels that would maximize the overall system performance under a power constraint. In [14], a neural network (NN)-based DVFS policy is proposed to optimize the temperature of the chip, by adjusting the V/f levels of the cores. Another work [8] used an NN to predict performance and power sensitivities of running applications. The predicted labels are combined to compute a boostability metric that guides the decisions of their DVFS policy, in order to maximize performance under temperature constraints on a manycore. These techniques have used the well-known RC thermal model [4] to predict the impact of adjusting V/f levels on temperature. However, such analytical models depend on the detailed chip floorplan and cooling system specifications, which are often unavailable at system level. To this end, more advanced SL-based thermal models [15] have been proposed and used by optimization techniques at runtime [16]. Nonetheless, these models are usually tailored for specific chips and environmental settings, e.g., ambient temperature, necessitating extensive re-training for new contexts. As demonstrated in [8], diverse training data improves the generalizability of SL-based models to unseen applications at runtime. Yet, generating training data for multiple chips and varying conditions is computationally prohibitive and impractical. As such, the literature lacks techniques that generalize to useen environment settings.

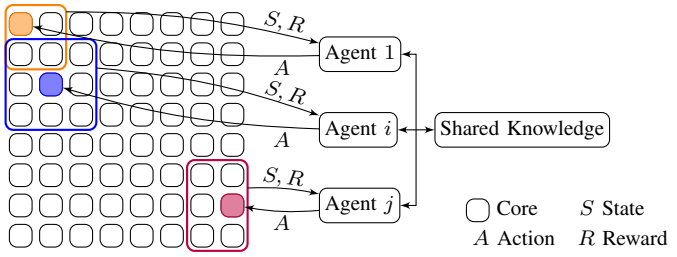

Fig. 1. An overview of our multi-agent RL for thermally-restricted performance optimization on manycores, where knowledge is shared among the agents.

To mitigate these shortcomings of SL, research has shifted towards the RL paradigm, with a primary focus on power/energy optimization. In [10] TQL-based DVFS is used to improve energy efficiency under performance constraints. A more advanced technique [9] used per-core multi-agent TQL-based DVFS to maximize the performance of manycores under a power constraint. Yet, fewer works in the literature have used RL for thermal optimization. DeadPool [17] used RL-based DVFS to minimize temperature under a performance constraint. A more recent work [12] used DQL to adjust the V/f levels of the cores in order to minimize temperature under timing constraints. Both of these techniques do not enforce a temperature constraint. This is more complex than enforcing performance constraints, because of the thermal coupling between the cores, thereby DVFS decisions on a core might violate the temperature constraint on other cores. The work in [18] has aimed at enforcing a thermal constraint on the cores through TQL. However, this policy does not consider any knowledge about the running applications' performance, such as instructions per second (IPS), or Cycles per instruction (CPI). Thus, the objective of performance maximization is well achieved. In summary, none of these thermal optimization techniques has targeted to maximize application performance under a temperature constraint, nor did they target manycores. In contrast, our proposed technique aims at maximizing performance under a temperature constraint, while targeting many-core processors with multi-agent RL. The closest work [9] to our approach uses multi-agent RL to maximize performance for manycores under a power constraint, instead of temperature. In summary, *none of the state-of-the-art techniques has employed RL to maximize performance of manycores under a temperature constraint.*

III. Multi-Agent RL for DVFS on Manycores

This work considers a symmetric many-core processor with n cores and k V/f levels. Per-core DVFS is supported, similar to many recent commercial processors, e.g., [2]. The workload on the processor can involve a priori unknown applications that arrive at unknown times. The objective of this work is to maximize the overall system performance under a thermal constraint, T_{crit} for manycores. We employ RL to achieve this objective while adapting to varying environment settings: workloads, system utilization and ambient temperature. RL involves an agent that interacts with the environment by observing its state and taking actions on it. The agent learns by receiving a positive reward for good actions and a negative reward for bad ones. The environment in the context of this work is the manycore. The actions are the per-core V/f levels. The state includes relevant performance and temperature measurements.

Using one RL agent to accomplish the targeted optimization

is inadequate due to the large state and action spaces on manycores. For instance, the action space will involve all combinations of V/f levels for all cores, which is quantified as k^n. To provide a scalable approach with manycores, we employ multi-agent RL, where each agent is responsible for selecting the V/f levels of its core only. To accelerate the learning process, we enable *knowledge sharing* among the agents, which is one of the training schemes adopted in the theory of multi-agent RL [13]. Fig.1 shows an overview of our multi-agent RL that tackles the problem of thermally-restricted performance optimization on manycores via DVFS.

A. State/Action Space, and Reward Function

Considering the targeted objective, metrics for performance and temperature should be involved in the state. For performance evaluation, we add the current CPI and the V/f level of the core into the state. The CPI of a core can vary at runtime due to either a change in the execution phase of the running application or to a change in the V/f level performed by the core's agent. In particular, increasing the V/f level may negatively affect the CPI due to memory stalls. Hence, from these state components the agents can learn the impact of adjusting the V/f level on performance. Additionally, we consider the current temperature of the core as part of the state. However, the impact of changing the V/f on the core temperature is not instantaneous. Thus, we also consider the power of the core, as it will instantaneously change by V/f changes. Furthermore, due to the heat coupling between the cores, an agent's decision of upscaling the V/f level might lead to a thermal violation on neighboring cores. To consider this, we also include the maximum and the average temperature of the neighboring cores into the state. Thus, the state space consists of six components with continuous values, while the action space consists of k discrete actions, corresponding to the available V/f levels.

The reward function should reflect the optimization goal. Thus, an agent should be rewarded if it improves the performance of the running application without violating T_{crit} and penalized if T_{crit} is violated. While the metrics of CPI or IPS can be used to quantify the performance, they might be misleading for the agent. The reason is that their values may increase due to a change in the execution phase of the running application, even if the agent has downscaled the V/f level in the previous step. To avoid this ambiguity, we instead consider the frequency in the reward; if the agent manages to increase the frequency without violating T_{crit}, it is rewarded. In case of a thermal violation, the agent gets a negative reward value proportional to the magnitude of the violation ΔT, i.e., the difference between the current core temperature and T_{crit}. Eq. (1) shows the reward function, where f_{core} is the current core frequency. k_1 and k_2 are constants obtained empirically.

$$R = k_1 \cdot f_{core} - k_2 \cdot \max(\Delta T, 0) \tag{1}$$

B. RL Agent Algorithms

An RL agent starts interacting with the environment by taking random actions and observing the resulting reward. This strategy is called *exploration*. When trained, the agent can pick the best action that yields the maximum reward according to its current knowledge. This strategy is called *exploitation*. To strike a good balance between exploration and exploitation, we use the *epsilon-greedy* approach. Specifically, at each step, the agent chooses a random action with a probability of ϵ, and the best possible action to its knowledge with a probability of $1-\epsilon$. The value of ϵ decays as the training progresses to ensure more exploration in the early training stages and to adjust toward more exploitation in the later stages.

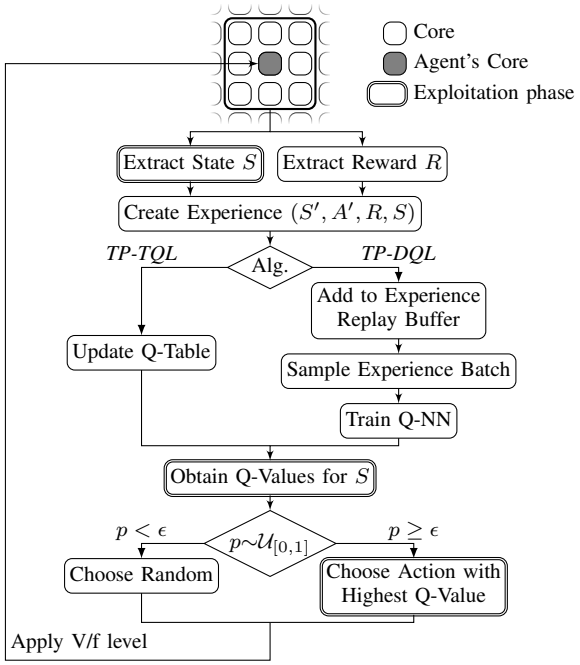

Fig. 2. The flow of the work of one agent in our proposed *TP-TQL* and *TP-DQL* including both exploration and exploitation phases.

The agent needs to estimate the quality (Q-value) of taking specific actions for specific states. There are two popular algorithms, namely TQL and DQL [13], that can perform this estimation. TQL uses a table, called Q-table, to store the Q-values of all available state-action pairs. It can handle only discete state/action spaces. For our RL agents, the action space is discrete, whereas the state components are continuous, hence they should be discretized. More fine-grained discretization can lead to more accurate learning, but it significantly increases the table size. DQL uses a fully-connected NN, called Q-NN, to estimate the Q-values of all actions considering a specific state. Hence, it accepts continuous-state values. Furthermore, the following two components are also used for DQL to improve its stability: 1) Experience Replay Buffer to store the experiences of the agents. Each experience corresponds to one training step and is represented by a tuple of the state S, the reward R, the previous state S', and the previous reward R'. At each training step, the Deep Q Networks (DQN) algorithm stores the current experience and samples a batch from the buffer to update the Q-NN. This mechanism breaks the correlation between consecutive states. 2) Target-NN, which is an additional NN to store the target Q-Values of a state. The Target-NN tackles the moving targets problem [13].

Importantly, DQL is superior to TQL [13], but is computationally more expensive, as it needs to train an NN during exploration and perform NN inference during exploitation. This is crucial for a DVFS policy as it is executed at each 1 ms. Therefore, we investigate the efficacy of both algorithms in

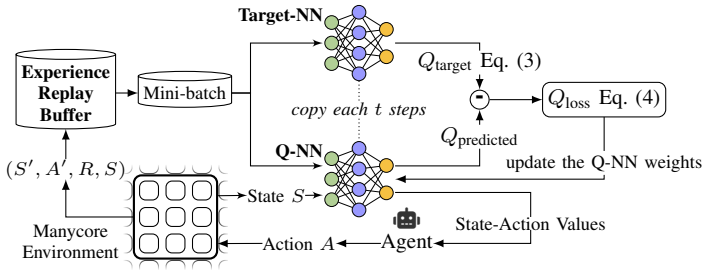

Fig. 3. The training step of *TP-DQL* using Target-NN and experience replay buffer to improve the quality of the learning.

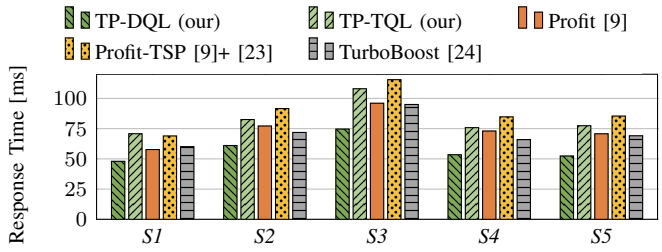

Fig. 4. Average response time of the applications in the workloads. *TP-DQL* consistently achieves a lower response time compared to other policies, with average and max improvements of 34.96% and 38.68%, respectively.

achieving our thermally-restricted performance optimization. We refer to our two policies as, *TP-TQL* and *TP-DQL*, and explain their flow in the following section.

C. *The flow of* TP-TQL *and* TP-DQL

Both our techniques are executed at each DVFS control epoch according to the flow shown in Fig. 2. Before starting the learning process, the Q-table is initialized with zeros, while the weights of the Q-NN are randomly initialized and copied to the Target-NN. To enable knowledge sharing among them, the agents use the same Q-table, Q-NN, Target-NN, and experience buffer. At each epoch and for each agent, the corresponding state S is extracted and the reward R is calculated (Eq. (1)) to evaluate the previous agent's action A' taken at the previous epoch considering the previous state S'. In case of *TP-TQL*, the cell in the Q-table that belongs to the previous state and reward, i.e., $Q(S', A')$, is updated according to the Bellman equation (Eq. (2).

$$Q(S', A') = Q(S', A') + \alpha \left[R + \gamma \max_{\hat{A}} Q(S, \hat{A}) \right] \quad (2)$$

In Eq. (2), γ refers to the discount factor that determines the importance of future rewards for the agent. $\hat{A}$ indicates the action that leads to the maximum Q-value at the current state S. α refers to the learning rate, which indicates the extent to which the newly acquired Q-values are updated. In the case of *TP-DQL*, a tuple of (S, R, S', R') is added to the experience buffer. Then, a mini-batch of experiences is sampled from the buffer to train the Q-NN as detailed in Fig. 3. First, each experience in the batch is passed to both Target-NN and Q-NN. Then, the target value Q_{target} is calculated according to Eq. (3) to compute the Q_{loss} as shown in Eq. (4).

$$Q_{\text{target}} = R + \gamma \cdot \max_{\hat{A}} Q(S, \hat{A}) \quad (3)$$

$$Q_{\text{loss}} = \text{MSE}(Q_{\text{predicted}} - Q_{\text{target}}) \quad (4)$$

The Q-NN is optimized through the known backpropagation process using a learning rate α that determines the amount of apportioned error with which the Q-NN weights are updated. The Target-NN is updated periodically at t intervals by copying the Q-NN weights to the Target-NN. Fig. 3 visualizes this training loop for Q-NN.

After updating the Q-table/Q-NN (depending on the employed algorithm), the agent either takes a random decision on the environment or selects the best action according to its Q-table/Q-NN. This is decided by comparing a random value between 0 and 1 with $epsilon$. After applying the selected V/f level, the loop starts again to extract the resulting new state and reward. The agents continue training and updating their Q-table/Q-NN until convergence. Once converged, they execute the exploitation steps shown in Fig. 2, to exploit their knowledge stored in Q-table/Q-NN and to select the best action for a given state. As to minimize the overhead of *TP-DQL* in these steps, the inference of the Q-NN for each b agents runs in a single batch on one core, where b is chosen empirically.

IV. Experimental Evaluation

To validate our proposed techniques, we conducted simulation experiments using *HotSniper* [19], an extended version of the state-of-the-art manycore simulator, *Sniper* [20], that incorporates periodic thermal simulations using *HotSpot* [4]. We simulated a symmetric 8x8 many-core processor, that is thermally restricted by 70°C. The default *HotSpot* cooling parameters are used. The experimental workloads consist of a combination of applications, randomly selected from the *PARSEC* [21] and *SPLASH-2* [22] benchmark suites. The frequency range is [1 GHz, 4 GHz]. At runtime, idle cores are set to the minimum V/f level, while the V/f levels of active cores are selected by the running DVFS policy, i.e., our *TP-TQL*, our *TP-DQL*, or the comparison techniques explained below.

Comparison Techniques: We compare our techniques against *Profit* [9], the closest technique to our work, as it uses multi-agent RL for performance maximization using DVFS. Nevertheless, their performance optimization is constrained by power instead of temperature. Therefore, we have implemented *Profit-TSP*, an augmented version of *Profit*, to restrict the optimization by the thermally-safe power constraint (TSP) [23], thereby making it thermally restricted. As TSP is a function of the number of active cores, we updated it at runtime according to the actual number of active cores. We also added the temperature to their reward function to penalize the agents for thermal violations. Additionally, we compare against the state-of-the-practice *Intel TurboBoost* [24], a simple control loop that checks the core temperatures each 1 ms, downscales the V/f levels of all cores by one step in the event of a thermal violation or upscales them otherwise. The DVFS epoch for all techniques is set to 1 ms, similar to commercial processors [2].

Evaluation Scenarios: As explained in Section III, during the early exploration phase, the agent takes random actions with high probability. Thus, its performance in this phase is not representative. Therefore, we bootstrap the agents in all comparison candidates (except for *TurboBoost*, which does not employ RL) by letting them explore the manycore while executing one initial workload and considering one ambient temperature of 45°C. The initial workload is repeated 120 times to have sufficient exploration, where the $epsilon$ decays linearly from 1.0 to 0.01 with a step of 0.025. This bootstrapping

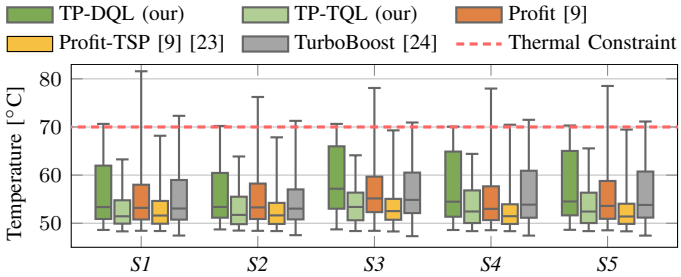

Fig. 5. The temperature distribution of the cores throughout the workload execution. *TP-DQL* keeps the maximum temperature right at T_{crit}, while resulting in higher temperatures compared to other policies, indicating its ability to leverage available thermal margins to gain performance.

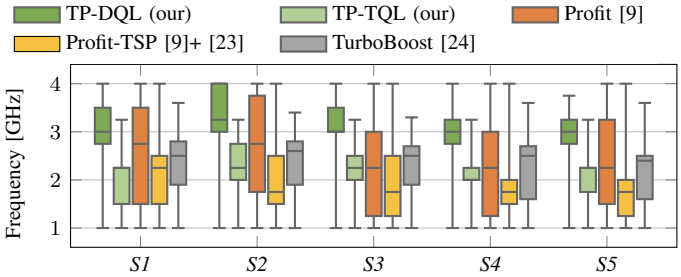

Fig. 6. Frequency distribution of the cores throughout the workload executions. *TP-DQL* sets the cores to higher frequencies in comparison to benchmark algorithms, leading to accelerated application execution.

phase takes up to 1 minute for the employed techniques.

We evaluate the performance of all techniques considering the following scenarios. In *S1*, 50% of the applications are unseen, i.e., not involved in the initial workload. In *S2*, 70% of the applications are unseen. These two scenarios adopt the same arrival rate of applications, i.e., 60 applications per second. The resulting system utilization, i.e., the ratio of active cores to idle cores, throughout the execution of each of these scenarios, ranges between 12.5% and 100% with an average of 51%. To test the techniques at a different system utilization ratio, we consider *S3*, which includes the same workload as *S2* but with a faster application arrival rate, i.e., 140 apps/s. Consequently, the resulting utilization in *S3* ranges between 37.5% and 100% with an average of 76%. In *S4*, 100% of the applications are unseen, that is, a completely new workload. The last scenario, *S5*, involves also a new workload, but this time with a different ambient temperature as well, i.e., 50°C. Hence, our evaluation scenarios involve changes in the workload, the utilization and the ambient temperature, to evaluate the ability of the RL-based techniques to adapt to environment changes at runtime.

As RL allows continual learning during runtime, the comparison techniques retrain their Q-table/Q-NN to adapt to the aforementioned environment changes. To this end, at the beginning of each evaluation scenario, the techniques reset $epsilon$ to its initial value 1.0, with a decay rate of 0.04. We have noticed that after running a workload for 20 seconds, the agents converge and do not gain any more knowledge. The resulting performance and temperature on the manycore following the agent actions after convergence are recorded for all techniques and will be discussed in the following.

Our RL Parameters: We have tuned the parameters of our RL agents empirically by experimenting with different values. The k_1 and k_2 constants of Eq. (1) are set to 1 and 5, respectively, considering the following argument. As k_1 determines the importance of the frequency in the reward, setting it to a higher value compared to k_2 might incentivize the agent to set the frequency to high levels at the cost of temperature violations. On the other hand, with a very large k_2 compared to k_1, the agent might be conservative and keep the core temperature very distant to T_{crit}, thereby missing on potential performance gains.

For the learning rate, α, we used the values of 0.1 and 0.0005 for *TP-TQL* and *TP-DQL*, respectively. The discount factor γ is set to 0.9 for both our techniques. These values result in a stable convergence in our experiments. For *TP-DQL*, both Q-NN and Target-NN have 2 hidden layers, each with 48 neurons and ReLU activation. The Q-NN is updated with an Adam optimizer. The target network is updated each 1000 optimization steps. The batch execution size b is set to 16, i.e., the Q-NN inference of each 16 agents runs as a single batch on one core. The experience replay buffer has a capacity of 100,000 experiences with a first-in-first-out replacement strategy. The Mini-batch size is set to 100.

Policy Performance: The primary metric determining the performance of the employed DVFS policies is the resulting overall system performance and core temperatures. In Fig. 4, we report the average response times of the applications in the workloads of our evaluation scenarios, as a metric for the overall system performance. Fig. 5 shows the quantiles of the temperatures of all cores in the system over the execution time, along with the min, the max and the median temperatures.

As shown in Fig. 4, *TP-DQL* stands out by achieving the best average performance of 34.96% among the five comparison techniques, i.e., the lowest average response time, while always satisfying the thermal constraint, as shown in Fig. 5. *TP-DQL* outperforms *TP-TQL* by 30.22% on average, which proves the superiority of deep learning in DQL over the simple TQL. As it can be seen on Fig. 5, *TP-TQL* does not exploit the available thermal headroom in maximizing the performance. This indicates that the agent in TQL was not able to gain the same knowledge as the DQL, although they are using the same reward function. This is expected due to the advancements in the DQL to enable more effective learning. Additionally, the discretization in TQL prevents the agent from distinguishing between states that have different continuous values, but the same discrete ones, thereby losing optimization potentials. The performance of our TQL is comparable with *Profit-TSP*, as both use the same RL algorithm, i.e., TQL. Nevertheless, our *TP-TQL* outperforms *Profit-TSP* by 6.72% on average and by up to 10.55%, because the latter uses TSP to enforce thermal safely, which is more conservative than directly enforcing T_{crit}.

It can be noticed that the resulting performance of *Profit* is higher than *Profit-TSP*, but at the cost of thermal violations (up to 10 °C). This is expected as *Profit* does not consider a thermal constraint. Interestingly, *TurboBoost* performs better than all TQL-based techniques. The reason is that *TurboBoost* oscillates around T_{crit} all the time. These slight violations of temperature allow for performance improvements. However, *TurboBoost* is designed to be applied for short periods of time [24] because continuous violations of temperature degrade reliability.

To gain deeper insights into the results, we report in Fig. 6 the quantiles of the frequencies of all cores that are selected by the DVFS policies throughout the execution time. As can be

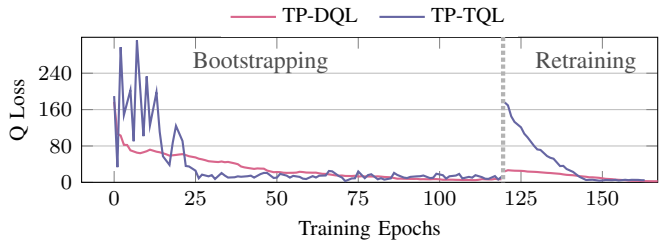

Fig. 7. Loss Progressions Throughout the Training to show the stability and adaptability of our techniques; *TP-TQL* and *TP-DQL*.

observed, *TP-DQL* exhibits the highest frequencies among all algorithms, thereby exploiting the full performance potential. These higher frequencies result in higher temperatures as well as shown in Fig. 5, but without leading to thermal violations. To evaluate the impact of randomness on *TP-DQL*'s performance, we run our evaluation scenarios with three bootstrapping processes (different random seed). The resulting average standard deviation is 1.6%, with a maximum value of 2.8%, indicating that *TP-DQL* is robust and minimally affected by randomness.

Stability and Adaptability of *TP-TQL* and *TP-DQL*: To demonstrate the ability of our RL techniques to converge after some exploration but still to adapt to environment changes, we examine the Q-loss values of our agents during the bootstrapping phase followed by the *S5* evaluation scenario, which exposes a completely new workload and a new ambient temperature. As shown in Fig. 7, the Q-losses have large values at the beginning of training, as the agents are exploring the environment. Then, the Q-loss starts decreasing, until convergence. When a change in the environment (arrival of a new workload or a change in the ambient temperature) is observed, the agents start to re-explore the environment. After a short period, i.e., 44 training epoch, the agents converge again.

Overhead Discussion: Since *TP-DQL* outperforms *TP-TQL* and requires more computation, we focus on calculating the computational overhead of *TP-DQL* in terms of time. For a bootstrapped DQL agent, the Q-NN update during the retraining phase takes 44 µs, i.e., 4% of the DVFS epoch. However, such retraining is not invoked at each DVFS epoch, but rather only following environment changes. The overhead associated with exploitation is of particular importance, as it recurs at each DVFS epoch. To accelerate it, we batch the Q-NN inference for each 16 agents on a single core. Hence, 4 cores out of 64 will execute the Q-NN inference at each DVFS epoch. This resulted in a per-core overhead of 2.4 µs per DVFS epoch, i.e., 0.24%, showing that *TP-DQL*'s computational overhead is negligible.

V. Conclusion

The paper introduces a novel self-adaptive and scalable performance optimization technique for thermally-restricted manycores, through the use of multi-agent RL. Our comparative analysis between two RL algorithms—TQL and DQL—reveals that DQL significantly outperforms TQL while incurring minimal overhead, thanks to effective knowledge sharing among agents. Our experimental evaluations show that our approach surpasses the state of the art by an average performance improvement of 34.96%, thereby affirming its capability to fully harness the system's performance potential within the predefined thermal constraint.

Acknowledgment: We would like to thank our former colleague Dr. Martin Rapp for the valuable discussions on this work.

References

[1] V. Narayanan and Y. Xie, "Reliability concerns in embedded system designs," *Computer*, vol. 39, no. 1, pp. 118–120, 2006.
[2] "Intel Corporation. Intel® core™ i9-12900kf processor," https://www.intel.com/content/www/us/en/products/sku/134600/intel-core-i912900kf-processor-30m-cache-up-to-5-20-ghz/specifications.html, 2021, (accessed: 2023-05-31).
[3] H. Wang, W. Li, W. Qi *et al.*, "Runtime Performance Optimization of 3-D Microprocessors in Dark Silicon," *IEEE Trans. Computers (TC)*, 2020.
[4] W. Huang, S. Ghosh, S. Velusamy *et al.*, "HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, 2006.
[5] M. Sagi, N. A. Vu Doan, N. Fasfous *et al.*, "Fine-grained power modeling of multicore processors using ffnns," *International Journal of Parallel Programming*, vol. 50, no. 2, pp. 243–266, 2022.
[6] M. Gupta, L. Bhargava, and I. Sreedevi, "Deep neural network learning for power limited heterogeneous system with workload classification," *Computing*, vol. 104, pp. 1–28, 01 2022.
[7] J. M. N. Abad and A. Soleimani, "Novel feature selection algorithm for thermal prediction model," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 10, pp. 1831–1844, 2018.
[8] M. Rapp, M. B. Sikal, H. Khdr, and J. Henkel, "Smartboost: Lightweight ml-driven boosting for thermally-constrained many-core processors," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 265–270.
[9] Z. Chen, D. Stamoulis, and D. Marculescu, "Profit: priority and power/performance optimization for many-core systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 10, pp. 2064–2075, 2017.
[10] E. Kwon, S. Han, Y. Park *et al.*, "Reinforcement learning-based power management policy for mobile device systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 10, pp. 4156–4169, 2021.
[11] S.-G. Yang, Y.-Y. Wang, D. Liu *et al.*, "Releta: Reinforcement learning for thermal-aware task allocation on multicore," *arXiv preprint arXiv:1912.00189*, 2019.
[12] D. Liu, S.-G. Yang, Z. He *et al.*, "Cartad: Compiler-assisted reinforcement learning for thermal-aware task scheduling and dvfs on multicores," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.
[13] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems*, G. Sukthankar and J. A. Rodriguez-Aguila, Eds. Cham: Springer International Publishing, 2017, pp. 66–83.
[14] H. M. Abbas, B. Halak, and M. Zwolinski, "Learning-based bti stress estimation and mitigation in multi-core processor systems," *Microprocessors and Microsystems*, vol. 81, p. 103713, 2021.
[15] K. Zhang, A. Guliani, S. Ogrenci-Memik *et al.*, "Machine learning-based temperature prediction for runtime thermal management across system components," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 2, pp. 405–419, 2018.
[16] L. Siddhu and P. R. Panda, "Predictncool: Leakage aware thermal management for 3d memories using a lightweight temperature predictor," *ACM Trans. Embed. Comput. Syst.*, vol. 18, no. 5s, oct 2019. [Online]. Available: https://doi.org/10.1145/3358208
[17] S. Dey, A. K. Singh, X. Wang, and K. D. McDonald-Maier, "Deadpool: Performance deadline based frequency pooling and thermal management agent in dvfs enabled mpsocs," in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, 2019, pp. 190–195.
[18] A. Iranfar, S. N. Shahsavani, M. Kamal, and A. Afzali-Kusha, "A heuristic machine learning-based algorithm for power and thermal management of heterogeneous mpsocs," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2015, pp. 291–296.
[19] A. Pathania and J. Henkel, "Hotsniper: Sniper-based toolchain for many-core thermal simulations in open systems," *IEEE Embedded Systems Letters*, vol. 11, no. 2, pp. 54–57, 2018.
[20] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–12.
[21] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Parallel Architectures and Compilation Techniques (PACT)*. ACM, 2008.
[22] S. C. Woo, M. Ohara, E. Torrie *et al.*, "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Int. Symp. Computer Architecture (ISCA)*, 1995.
[23] S. Pagani, H. Khdr, J.-J. Chen *et al.*, "Thermal safe power (tsp): Efficient power budgeting for heterogeneous manycore systems in dark silicon," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 147–162, 2016.
[24] *Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors*, Intel Corporation, 2008.