
Article:
Wang, Y. orcid.org/0000-0003-3298-5134, Hao, M. orcid.org/0000-0003-0043-4370, He, H. orcid.org/0000-0002-6494-775X et al. (4 more authors) (2024) DRLCap: Runtime GPU Frequency Capping with Deep Reinforcement Learning. IEEE Transactions on Sustainable Computing. ISSN 2377-3782

https://doi.org/10.1109/tsusc.2024.3362697

DRLCAP: Runtime GPU Frequency Capping with Deep Reinforcement Learning

Yiming Wang, Meng Hao, Hui He, Weizhe Zhang, *Senior Member, IEEE,* Qiuyuan Tang, Xiaoyang Sun, and Zheng Wang, *Member, IEEE*

***Abstract*—Power and energy consumption is the limiting factor of modern computing systems. As the GPU becomes a mainstream computing device, power management for GPUs becomes increasingly important. Current works focus on GPU kernel-level power management, with challenges in portability due to architecture-specific considerations. We present DRLCAP, a general runtime power management framework intended to support power management across various GPU architectures. It periodically monitors system-level information to dynamically detect program phase changes and model the workload and GPU system behavior. This elimination from kernel-specific constraints enhances adaptability and responsiveness. The framework leverages dynamic GPU frequency capping, which is the most widely used power knob, to control the power consumption. DRLCAP employs deep reinforcement learning (DRL) to adapt to the changing of program phases by automatically adjusting its power policy through online learning, aiming to reduce the GPU power consumption without significantly compromising the application performance. We evaluate DRLCAP on three NVIDIA and one AMD GPU architectures. Experimental results show that DRLCAP improves prior GPU power optimization strategies by a large margin. On average, it reduces the GPU energy consumption by 22% with less than 3% performance slowdown on NVIDIA GPUs. This translates to a 20% improvement in the energy efficiency measured by the energy-delay product (EDP) over the NVIDIA default GPU power management strategy. For the AMD GPU architecture, DRLCAP saves energy consumption by 10%, on average, with a 4% percentage loss, and improves energy efficiency by 8%.**

***Index Terms*—GPUs, deep reinforcement learning, power and energy optimization, GPU power optimization**

I. INTRODUCTION

GRAPHICS processing units (GPUs) are the mainstream computing devices on many computing systems. They are used to accelerate a wide range of tasks, from traditional high-performance applications [1], [2] to emerging workloads like deep learning models [3]. However, the high power consumption of GPUs significantly impacts their reliability, economic viability and operational cost. This is a particular issue for GPU clouds and high-performance computing (HPC) systems, where the GPU power and cooling infrastructures contribute to a large part of the operational cost [4]. As a result, there is a critical need to reduce GPU power consumption without significantly compromising the application performance.

Power management can be achieved through frequency capping on commercial off-the-shelf GPUs. This is done by limiting the maximum GPU clock frequency to implicitly control the GPU power consumption [5], [6]. There is a wide range of power management schemes proposed in the CPU space to exploit dynamic voltage and frequency scaling (DVFS) through e.g., machine learning [7], [8], expert-crafted heuristics [9]–[12] or analytical models [13]–[15]. However, many of the CPU-specific techniques are not transferable to the GPUs because of the complex non-linear relationship between processor frequency and application performance. Additionally, several studies [16]–[19] utilize overclocking and undervolting to conserve power in HPC. However, these works may potentially introduce faults. As such, it remains an outstanding challenge to trade application performance for energy efficiency on GPUs.

Most recent works on GPU energy rely on profiling information at the kernel level of the application for the targeted GPU architecture [20], [21]. Such a strategy can incur significant profiling overhead or necessitate the instrumentation of the GPU kernel, and it cannot easily adapt to the changing workload behavior resulting from dynamic inputs (which are often hard to anticipate ahead of time). Some other works attempt to choose a power setting during program execution time [22], [23], but they focus on iterative workloads, such as Convolution Neural Networks training. As such, these approaches are ineffective if there are dynamically changing behaviors across program phases during an iterative workload, e.g., moving from a computation-intensive kernel to a memory-intensive kernel where the optimal power configuration is likely to be different. In addition, these works at the kernel level face challenges in portability to GPUs with different architectures due to variations in instruction sets, memory hierarchies, and hardware features inherent to each architecture.

To address these challenges, this paper presents DRLCAP, a general GPU power management system designed to optimize GPU power consumption through dynamic GPU frequency capping (and memory capping if the underlying architecture supports it, or power capping if the architecture supports very few frequency caps). One of the key challenges in dynamic power management is detecting GPU workload changes and adapting to such changes. DRLCAP addresses this challenge by adopting an application-transparent, low-overhead

Yiming Wang, Meng Hao, Hui He and Weizhe Zhang are with the School of Cyberspace Science, Harbin Institute of Technology, Harbin 150001, China. E-mail:{yimingw, haomeng, hehui, wzzhang}@hit.edu.cn.

Qiuyuan Tang is with the Bili Bili Technology Co., Ltd. Shanghai, China. E-mail:tangqiuyuan@bilibili.com.

Xiaoyang Sun and Zheng Wang are with the School of Computing, University of Leeds, LS29JT Leeds, U.K. E-mail: {scxs, z.wang5}@leeds.ac.uk.

Manuscript received XX XX, 2023; revised XX XX, 2023.

and system-level profiling mechanism that monitors the GPU core and memory utilization and runtime power readings. This approach utilizes system-level information, making it more portable and eliminating the need for any kernel-level information. It then leverages the recent advance of deep reinforcement learning (DRL) to develop a runtime optimization framework to adjust and apply frequency capping based on the runtime system-level information. Built on DRL, DRLCAP learns directly from raw data without the need for manual feature extraction. It autonomously develops an adaptive power policy through interaction with the GPU system, enhancing its portability. DRLCAP is decentralized, where we can employ a DRL agent for each GPU in multi-GPU systems to monitor the workload and manage power on the local GPU.

We formulate the problem of finding the optimal GPU frequency setting as a single-player game. In this game, the player (DRLCAP) chooses a set of frequency settings given the current system status, aiming to improve the overall energy efficiency of the system. The goal of DRLCAP is to minimize the GPU energy consumption without significantly compromising the performance by dynamically changing the GPU frequency gap based on the current program status (or phase). DRLCAP does not only focus on immediate GPU energy efficiency but also considers the integrated impact of potential future GPU energy efficiency.

Compared to traditional supervised-learning methods [23]–[25], DRLCAP does not require an extensive training dataset, which is expensive to collect for program optimization due to the overhead of profiling. DRLCAP addresses the scarcity of data and minimizes the runtime overhead by combining offline and online learning. It is first trained offline on training benchmarks, and the trained network can then be applied to any new, unseen applications. As the policy network of DRLCAP also gets updated during runtime, the optimization strategy is gradually tailored for the target program under a specific input. Such a strategy is useful for long-running GPU applications where power consumption is more likely to be an issue than short-running ones. As we will show later in the paper, the use of a deep learning policy network and DRLCAP's ability to continuously learn and adapt lead to better performance for dynamic GPU power optimization than prior work.

We have implemented a prototype of DRLCAP[1]. We evaluate DRLCAP by applying it to an NVIDIA Tesla V100S. To verify the hardware portability, we also apply DRLCAP to NVIDIA RTX 3080 Ti, NVIDIA RTX 2080 Ti and AMD Radeon RX 6700 XT GPUs. We compare DRLCAP against four alternative schemes: a state-of-the-art GPU power coordination approach [26], our previous RL-based power optimization system designed for multicore CPUs [7], and two implementation variants that rely on power capping and a combination of frequency and power cappings. Experimental results show that DRLCAP consistently outperforms the alternative scheme when optimizing for GPU energy efficiency measured by the energy-delay product (EDP). On average, it reduces GPU energy consumption by 22% with a modest 3% slowdown in the application execution time on three NVIDIA GPUs compared to the default GPU power management scheme. For AMD GPU, the average energy saving and percentage loss are 10% and 4% respectively. Additionally, our decentralized decision-making is scalable and adaptable to larger distributed environments with multiple GPUs.

This paper makes the following contributions:

- It presents the first universal DRL framework for dynamic GPU power and energy optimization, autonomously discovering a near-optimal power-performance tradeoff controlled by a user-defined parameter (Section IV-A);
- It shows how a low-overhead and system-level approach can be integrated with DRL to detect changes in program behavior, enabling portability across different GPU architectures (Section IV-B);
- It demonstrates how offline and online learning can be combined in a DRL framework for dynamic GPU power optimization, enhancing the adaptability and efficiency of DRLCAP (Section IV-D);
- It evaluates DRLCAP across both NVIDIA and AMD GPUs, providing an assessment of its performance and adaptability on different architectures (Section VI-D).

II. Background and Related Work

A. GPU Power Optimization

Modern GPU architectures provide a large number of processing units and computation sources like specialized cores for floating-point and integer numbers as well as register files and share memory and caches. Depending on the application workloads, not all these computation resources are fully utilized at any given time. This provides an opportunity for a runtime system to dynamically limit the maximum GPU frequency for energy optimization.

Our work is a general-purpose GPU energy efficiency optimization solution. It not only can be applied to NVIDIA GPU architectures but also supports AMD GPU architectures. In this work, we evaluate our techniques on the NVIDIA and AMD GPU architectures. GPU supports two power management knobs: power capping and frequency capping. Power capping automatically recovers the unused power budget and moves it to another component, e.g., from memory to cores. By contrast, frequency capping is finer-grained since we can set different frequencies for the GPU core and memory, where the memory frequency can be adjusted independently of the core frequency by DVFS. In this paper, we consider dynamically tuning the GPU frequency cap to improve GPU energy efficiency. Thus, we compare the frequency capping scheme with the power capping scheme and the hybrid scheme of frequency and power capping in Section VI-B. Note, there are very few frequencies that can be manually adjusted on AMD GPUs, such as RX 6700 XT and hence on the AMD GPU, we only perform power capping. We use NVIDIA System Management Interface (nvidia-smi) and Pynvml to enforce the NVIDIA GPU caps for the power and frequency respectively. Both tools are based on NVIDIA Management Library (NVML). The user invokes NVML API functions, prompting the GPU

[1]Code and data will be released at https://github.com/yiminga/DRLCap upon acceptance.

driver to send corresponding instructions to the GPU, thereby achieving the intended functionality. For instance, the NVML function *nvmlDeviceSetApplicationsClocks* can be employed to dynamically adapt GPU clock frequencies. We use ROCm System Management Interface Library (rocm-smi) to control AMD GPU power and frequency cap.

B. Deep Reinforcement Learning

Reinforcement learning (RL) is a framework in which an agent interacts with the environment to learn a policy. The agent observes the state s_t, takes an action a_t at timestep t, immediately receives a reward r_t, and the environment transfers to the next state s_{t+1}. The goal of the agent is to maximize the cumulative reward over time. The action value function (Q-value) is a measure of the expected cumulative reward when taking a specific action in a given state.

Recently, deep reinforcement learning approximates the Q-value function using a deep neural network (DNN), addressing the challenge of processing Q-values in traditional reinforcement learning. However, using a DNN has been shown to be prone to instability or even divergence. To enhance stability, prioritized experience replay and target networks were introduced, that is prioritized experience replay Double Deep Q-network (DDQN) [27], [28]. The prioritized DDNN is updated sampling a mini-batch from experience replay buffer storing state transitions. Transitions with higher priority are sampled more frequently, leading to improved sample efficiency and faster convergence. Moreover, by decoupling the target and online networks, DRL uses a separate DDNN that is periodically updated to approximate the Q-values used in the target calculation. This helps mitigate the issues of overestimation bias and enhances the stability of the training process.

The DRL agent can be learned through combining offline training with online deep Q-learning. During the offline training phase, the agent accumulates a substantial dataset of state-action pairs (s, a) through interactions with the environment. This dataset is used to construct an initial DNN. Subsequently, in the online deep Q-learning phase, the agent continuously refines its policy through real-time interactions with the environment, updating the DNN parameters to adapt to dynamic changes and improve decision-making. In this work, we propose a DRL framework that combines offline training with online deep Q-learning, enhancing learning efficiency by leveraging a substantial dataset during initial training and continuously adapting to real-time interactions for improved decision-making.

C. Related Work

Our work builds upon the following previous foundations, but quality differs from each.

Dynamic power management. The majority of previous efforts in dynamic power and performance optimization have been focused on multicore CPUs [7], [10], [11], [15]. For example, our prior work on multi-core systems [7] employs a reinforcement algorithm, Double Q-learning, which involves maintaining two state-action tables. However, when attempting to port this CPU-specific approach to GPUs, as demonstrated in Section VI-B, the results were not satisfactory. Existing efforts [20], [29]–[31] on GPU power management mainly is GPU kernel-level performance and power modeling. Arafa et al. [29] utilized instrumentation tools to capture the behavior of the kernels to predict GPU performance. Kandiah et al. [30] utilized Parallel Thread Execution (PTX) and Source And Assembly (SASS) for GPU power modeling. COORD [26] is a static category-based heuristic for adjusting the GPU core and memory power. It coordinates the power allocation between the GPU core and GPU memory to balance computation and memory access. It requires knowing if the target program is compute- or memory-bound ahead of time. We compare our approach against COORD in Section VI-B. Several works [22], [23], [23], [32], [33] have employed power and performance models to proactively optimize for the present timestep by anticipating future occurrences to maximize the energy efficiency. However, these works focus on iterative or periodic applications and ignore future kernel behavior. In contrast to prior approaches, our approach is efficient for non-iterative programs on GPUs and focuses on incorporating long-term returns. Several works [34]–[39] assign multiple tasks with heuristic scheduling algorithms to improve energy efficiency on heterogeneous systems. Our DRL-based approach automatically adapts to the dynamic environment without an expert-crafted mechanism.

DRL-based power optimization. Machine learnins based power managers [7], [40]–[43] learn to improve their judgments over time by leveraging feedback from the system environment. PowerCoord [42] coordinates the power budget among the various power domains using reinforcement learning. However, this work needs prior knowledge of the application behavior. For example, job deadlines must be collected ahead of time for each task. The work [43] utilizes reinforcement learning to improve the energy efficiency of the field programmable gate array (FPGA). The work [41] provides a Double-Q power management approach to scale operating frequency for mobile devices. Our previous work [7] also used double Q-learning to adjust CPU power configuration to reduce the power consumption of multicore systems. As we have shown in Section VI-B, however, this approach fails to save more energy. Thus, to save more GPU energy, DRLCAP employs double deep Q-network with prioritized experience replay instead of double q-learning to dynamically scale the GPU frequency, which can adapt to the high dimensional input vector and avoid large-size storage.

III. MOTIVATION

As a motivation example, we consider applying GPU frequency and power capping for energy optimization on an NVIDIA V100S. On this platform, the GPU core frequency cap is modulated from 1597 MHz (default) to 135MHz (min) in steps of 150 MHz. The memory frequency cap supports a static setting of 1107 MHz. The GPU power cap is modulated from 250 W (default) to 100 W (min) in steps of 15 W. In this experiment, we apply frequency and power cappings to four

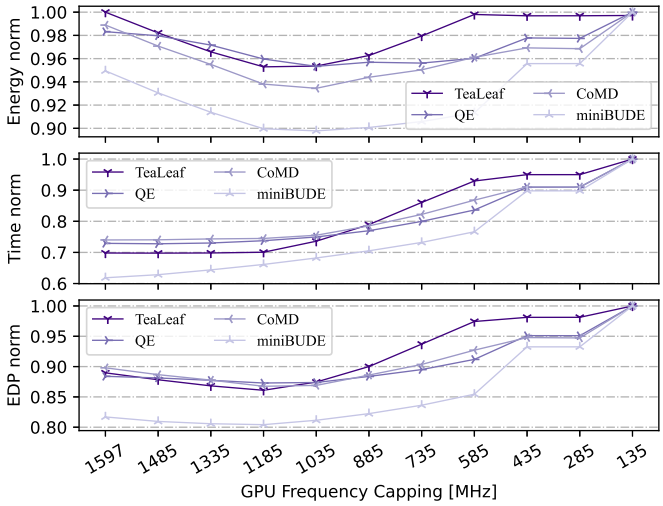

Fig. 1. Impact of GPU frequency capping on NVIDIA V100S across benchmarks.

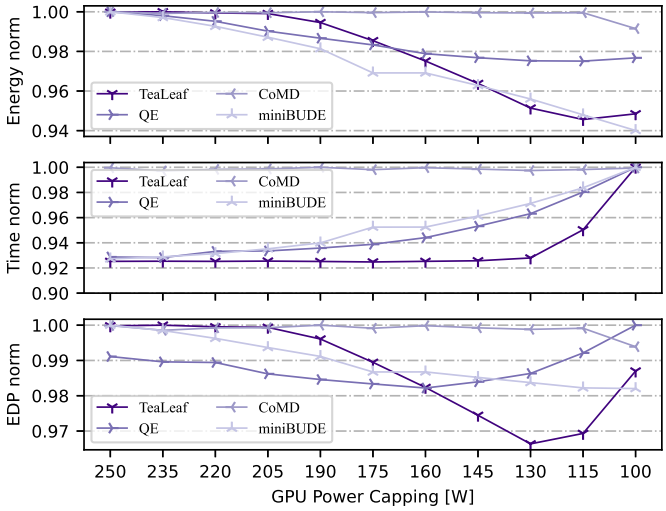

Fig. 2. Impacts of GPU power capping on NVIDIA V100S across benchmarks.

benchmarks with diverse workload characteristics: TeaLeaf [44], miniBUDE [45], CoMD [46] and Quantum ESPRESSO (QE) [47]. TeaLeaf and CoMD are memory-bound programs, miniBUDE is compute-bound, and Quantum ESPRESSO switches between multiple memory- and compute-bounded phases. In the evaluation, we report the measured program execution time (in seconds), energy consumption (in Joules), and the energy-delay product (EDP) - computed by multiplying the execution time with the energy consumption. The EDP is a lower-is-better metric that measures the trade-off between performance and energy consumption and is a widely used metric for quantifying energy efficiency [48]. We scale the number into the [0,1] range using a log function.

Figs. 1 and 2 report the normalized energy saving, runtime slowdown and EDP (with respect to running the application with the maximum frequency) when running an application under different GPU core frequency caps and power caps respectively. Reducing the GPU core frequency cap can have a significant impact on the execution time, but a lower frequency cap does not always lead to lower energy consumption as a

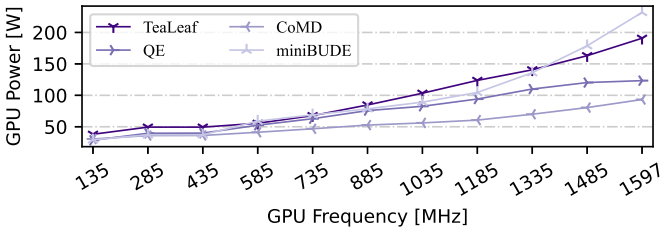

Fig. 3. Frequency-wattage curve on NVIDIA V100S across benchmarks.

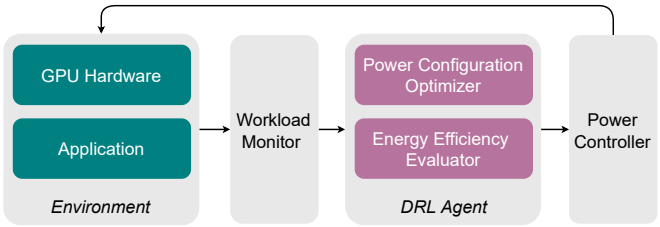

Fig. 4. Overview of DRLCAP framework.

longer execution time can result in more energy consumption. If we look at power capping in Fig. 2, we see that the energy consumption decreases and runtime increases as we lower the maximum power consumption. However, reducing the power cap has few impacts on the execution time of the memory-bound program CoMD. By contrast, power capping has a greater impact on the execution time for compute-bound programs like miniBUDE. We also observe the sweat spot of EDP (lower is better) varies across benchmarks.

The results show that the optimal frequency and power cap setting can vary across programs. For example, while QE achieves the minimum energy consumption at 1035 MHz, the performance loss is 13% compared with the default (1597 MHz). Thus, finding the right trade-off between energy consumption and performance is non-trivial. DRLCAP is designed to efficiently trade performance for energy consumption through DRL and dynamic adaptation. Moreover, as illustrated in Fig 3, the connection between GPU frequency and power consumption typically exhibits nonlinearity. This nonlinearity stems from a multitude of factors that impact GPU power usage, including voltage, temperature, and power management mechanisms. For instance, as frequency escalates, the requirement for higher voltages to maintain stability results in an exponential surge in power consumption. Hence, conventional heuristic approaches may encounter difficulties in uncovering the intricate non-linear connection between GPU frequency and power consumption. In contrast, our reinforcement learning model can reveal this relationship and identify the ideal equilibrium for frequency settings, aiming to maximize performance while concurrently minimizing power consumption.

IV. Our Approach

A. *Overview of* DRLCAP

DRLCAP is a general runtime system that automatically manages the GPU power configuration by interacting with the environment. It aims to reduce GPU energy consumption without dramatically compromising performance.

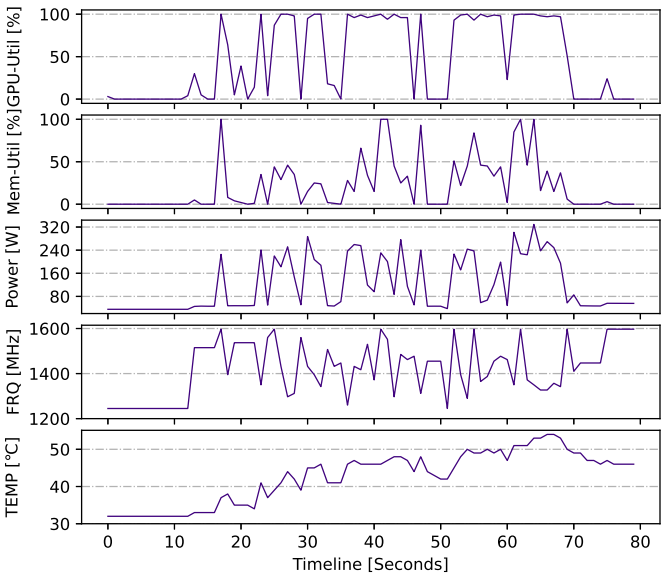

Fig. 5. Workload changes in Quantum ESPRESSO.

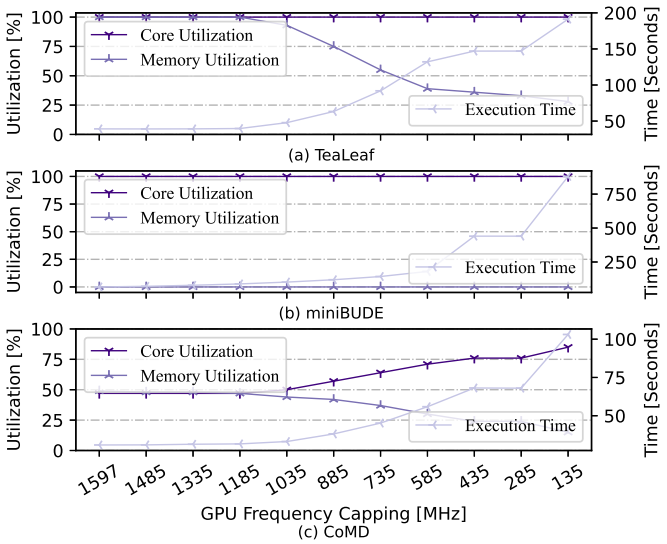

Fig. 6. Impact of GPU frequency capping on GPU hardware utilization and performance.

As depicted in Fig. 4, DRLCAP consists of four components. The workload monitor measures the workload characteristics of GPU applications through lightweight system-level profiling information. It uses hardware resource utilization and power traces to detect phase changes in the GPU workload. This process is entirely transparent to applications. The runtime measurements are sent to the DRL agent for runtime power management. The DRL framework consists of two components, an energy efficiency evaluator and a power configuration optimizer. The energy efficiency evaluator quantifies the quality of the current power configuration by considering the change in GPU power consumption and frequency (Section IV-C) for the current phase. Based on the evaluation, the power configuration optimizer then chooses a power configuration for the current state. The power configuration is adjusted by GPU frequency capping. Note that DRLCAP is a decentralized system that applies these four components on each GPU in a distributed environment.

B. Detecting Phase Changes

DRLCAP uses DRL to choose a GPU power setting according to the current environment state, aiming to maximize the cumulative reward. Specifically, in DRLCAP, the environment state is represented by a vector of five numerical values, including *GPU core utilization*, *GPU memory utilization*, *power*, *GPU frequency*, and *temperature*. Our rationale for using these metrics is described as follows.

These metrics obtained through low-cost system-level profiling do not necessitate any modifications to the source code of applications, nor do they require instrumentation of the GPU kernels. Additionally, many applications exhibit two distinct phases, compute-bound and memory-bound. For the compute-bound phase, a commonly used performance metric is the GPU core utilization. In this work, we compute the GPU utilization by measuring the percentage of time that one or more kernels were running on the GPU during the sampling period. A high GPU core utilization rate indicates that the kernels spend most of their time on GPU processing. Moreover, we use memory utilization to assess how frequently a GPU kernel accesses the global (device) memory to model the behavior of memory-bound workloads. Memory utilization is computed as the percentage of time that the GPU global memory was read or written during a sampling period. Since the number of memory accesses each period increases as the memory utilization rate rises, the kernel workload becomes increasingly memory-bound. Like the hardware utilization, the real-time frequency also provides similar contributions. Last but not least, power consumption and temperature are direct power indicators.

Fig. 5 shows the hardware utilization, power profile, current frequency and temperature of the Quantum ESPRESSO benchmark. We can observe several phases with distinct measurements across the metrics. By monitoring these metrics, DRLCAP can model the state of applications and the GPU. For example, we can observe long compute-bound phases characterized by high GPU core utilization, high power and high GPU frequency. The phase change can be detected by GPU hardware state information.

C. Evaluate the Energy Efficiency

Reinforcement learning is an online learning algorithm that can adjust and adapt its decision over time using feedback from the environment. For each state, the RL agent takes action and uses a feedback measurement to evaluate the quality of the actions taken so far. The feedback is given by a reward function that takes the change of GPU power consumption and frequency measured at runtime, the GPU core utilization and memory utilization as input and produces a single instantaneous reward value.

1) Reward function: We propose a heuristic method to build a reward function to trade off performance and energy consumption to maximize energy efficiency. In this work, we use EDP to model the trade-off between performance and energy consumption, but other metrics can be used too. Our

Algorithm 1: Heuristic Reward Function

Input: power consumption p, frequency f, core utilization u_{core}, memory utilization u_{mem}
Output: Reward $r(s,a)$

1 Initialize $p_1 < p_2 < p_3,\ r_1 > r_2 > r_3 > r_4$;
2 Take action a;
3 Observe next power consumption p', frequency f';
4 $\triangle p = p' - p$, $\triangle f = f' - f$;
5 **if** $u_{mem} == 0$ **then**
6 **if** $u_{core} == 100$ **then** ▷ case A
7 Set frequency caps to the max value;
8 Measure the power consumption p';
9 $\triangle p = p' - p$;
10 **if** $\triangle p \leq p_1$ **then** $r(s,a) = r_1$;
11 **elseif** $p_1 < \triangle p \leq p_2$ **then** $r(s,a) = r_2$;
12 **elseif** $p_2 < \triangle p \leq p_3$ **then** $r(s,a) = r_3$;
13 **else** $r(s,a) = r_4$;
14 **end**
15 **else if** $u_{core} == 0$ **then** ▷ case B
16 $r(s,a) = 1/action * c$;
17 **end**
18 **else** ▷ case C
19 Compute score g based on $\triangle p$ and $\triangle f$;
20 $r(s,a) = g/u_{core}$;
21 **end**
22 **end**
23 **else** ▷ case D
24 Compute score g based on $\triangle p$ and $\triangle f$;
25 $r(s,a) = g \times u_{mem}$;
26 **end**

reward function is designed to maximize the EDP. Here, we use the GPU utilization as the performance indicator and the energy reading to compute the EDP.

Fig. 6 shows GPU core and memory utilization and program execution time for three representative programs running at different GPU frequencies. Here, the x-axis gives different GPU frequency settings. We can observe that lowering the GPU frequency does not necessarily lead to a slowdown in the program execution time. Slowdown only starts occurring when the GPU frequency is lower than a certain threshold, and the threshold varies across programs. We also observe a strong correlation between runtime and memory utilization. In general, a lower memory utilization leads to longer execution times. This is because it takes longer to access the GPU memory under a lower frequency, leading to fewer memory accesses with a given sampling period and low memory utilization. We also observe that the GPU core utilization varies across programs. For Tealeaf and miniBUDE, the core utilization is largely independent of the GPU frequency, while for the memory-intensive CoMD benchmark, the core utilization is low with a high frequency. Our reward function is designed to model these program characteristics using low-cost runtime measurements.

Algorithm 1 outlines our reward function. To compute the reward, we first check the GPU utilization, power and frequency. We then consider four scenarios based on the utilization of the GPU core and memory.

A. For highly computation-intensive cases, we set the frequency capping to the maximum value. We then check the power consumption again and compute the reward value according to the change in the power consumption (lines 10 to 13 in Algorithm 1).
B. When the GPU is idle (e.g., no GPU kernel is running), we compute the reward as the reciprocal of action to minimize the frequency.
C. For other non-memory access cases, we compute the change of power and frequency to give a score g to reduce power consumption while maintaining the frequency. For this case, the reward is equal to this score divided by the core utilization.
D. Finally, for the remaining scenarios, we compute the change of power and frequency to get a score g, and then the reward is equal to this score multiplied by the memory utilization.

To compute the score g in cases C and D based on the changes in power and frequency, the steps are as follows. Let $\triangle power$ and $\triangle freq$ be the change in GPU power consumption and GPU frequency, respectively, during the current and the previous sample period.

a. If $\triangle power$ is less than or equal to zero, it indicates a decline and stabilization in GPU power, respectively. For these scenarios, our reward function should give a high instantaneous score. To minimize the impact on the application performance, we also compare the change of GPU core frequency, $\triangle freq$. We use the well-known clock and power formula, $P \propto FV^2$, to model the relation between GPU power and frequency. Here, F is the GPU clock frequency and V is the GPU supply voltage. Since that power and frequency are positively correlated, $\triangle freq$ is generally less than or equal to zero when $\triangle power$ is less than or equal to zero. This change in frequency indicates a decrease in GPU activity, but the tradeoff between power consumption and clock frequency can be non-trivial. In this work, we leverage the piecewise linear regression method to assign scores at a fine-grained level. The $\triangle freq$ is divided into different levels. The lower the value of $\triangle freq$, the greater the score is set. Therefore, when $\triangle power$ is less than or equal to zero, DRLCAP obtains scores g by the piecewise linear regression function as follows:

$$g(s,a) = \begin{cases} g_1 & -n \times step \leq \triangle freq \\ g_2 & -2 \times n \times step \leq \triangle freq < -n \times step \\ g_3 & -3 \times n \times step \leq \triangle freq < -2 \times n \times step \\ g_4 & \triangle freq < -3 \times n \times step \end{cases} \tag{1}$$

where $g_1 > g_2 > g_3 > g_4$ are the numerical scores that determine how much the magnitude of reward and penalty, n is a positive integer, and $step$ is the step of GPU frequency depending on the hardware environment. For example, in this work, we set $step$ to a multiple of 15 on our NVIDIA V100S GPU because of a step of 7 MHz alternating with 8 MHz on this GPU architecture.

Algorithm 2: D

Input: GPU a

Output: Actio

1 Initialize Iterat
exponents α
$sumtree = \emptyset$
Q with rando
function Q^-

2 Execute GPU

3 **for** $t = 1, 2, ..$

4 Observe G

5 With proba

6 With proba
$a = \text{argm}$

7 Set GPU p
capping a

8 Observe re

9 Store trans
maximal

10 Sample mi
transitions
based pric

11 Compute i
$w_j = (N$

12 Compute t
$y_j = \begin{cases} r \\ r \end{cases}$

13 Compute T
$\delta_j = \frac{1}{k} \sum$

14 Update trai

15 Update we

16 Every C st
$\theta^- \leftarrow \theta$;

17 **end**

b. If $\triangle power$
GPU power,
score. We t
higher than
in CPU acti
score. When
obtains scor
similar to th

D. Implementatic

Our power opti
experience repla
Algorithm 2. We
the double deep
train the prioritize
to the following

The workload monitor periodically measures workload characteristics and uses the measurement to detect workload changes. These workload characteristics will be preprocessed and discretized into a state set s (line 4 in Algorithm 1). For the current state s, the action with the maximum value is chosen as the optimal power configuration by the $\epsilon - greedy$ strategy, where $0 < \epsilon < 1$ is a parameter that controls how much exploration vs. exploitation is done (lines 5 and 6 in Algorithm 1). Exploration is chosen with probability ϵ, and action is selected uniformly at random. Alternatively, exploitation is chosen with probability $1 - \epsilon$, and the agent selects the action that it considers will have the best long-term consequence (ties between actions are broken uniformly at random). During the offline training stage, we update ϵ following a schedule to make the agent explore progressively less. However, during the online deployment stage, we fix ϵ the best-performing value found during offline training.

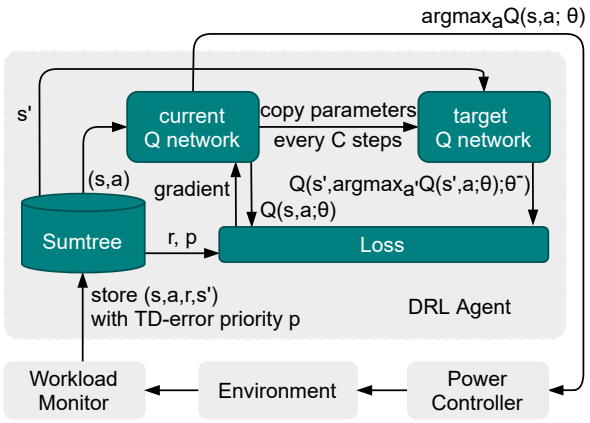

Fig. 7. Architecture of our power configuration optimizer.

The energy efficiency evaluator computes the reward r for the current power configuration taken in a given state (line 8 in Algorithm 1). As shown in Fig. 7, to perform experience replay, we store the experience transitions (current state s, action a, reward r, next state $s\prime$) in $sumtree$, a data structure in which each node is the sum of its children, with the leaf nodes serving as the priorities (line 9 in Algorithm 1), to replay critical transitions more frequently, and therefore learn more efficiently. The priority of transitions is measured by the magnitude of their temporal-difference (TD) error (lines 13 and 14 in Algorithm 1). New transitions arrive with no known TD error, thus we prioritize them to ensure that every experience is observed at least once.

This algorithm then leverages the stochastic sampling method to choose k transitions to avoid the loss of diversity (line 10 in Algorithm 1). The probability of sampling transition j as $P(j) = p_j^\alpha / \sum_i p_i^\alpha$, where α determines how much prioritization is used, with $= 0$ corresponding to the uniform case. Furthermore, this algorithm corrects the bias introduced by prioritized replay by using importance-sampling weights $w_j = (1/(N \cdot P(j)))^\beta$, which fully compensates for the non-uniform probabilities $P(j)$ if $\beta = 1$. For stability reasons, this algorithm normalizes weights by $1/max_i w_i$, so that they only scale the update downwards (line 11 in Algorithm 1).

According to the environmental state representation, the

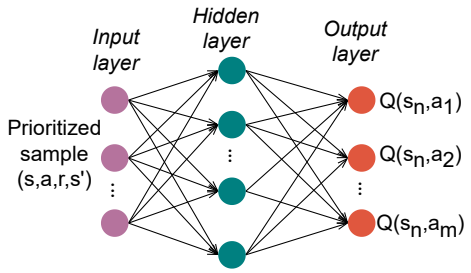

Fig. 8. Three-layer fully connected neural network model.

DRL agent selects the GPU power configuration with the biggest Q value according to the optimal action-value (also known as Q) function. We parameterize an approximate value function $Q(s, a)$ using the neural network. The agent uses two asynchronous neural networks, Q and Q^-, with the same structure and parameters, as shown in Fig. 8. In our work, the fully connected neural network (FCNN) is applied to both the current network and the target network. The extracted prioritized samples $(s,\ a,\ r,\ s\prime)$ are used as the inputs. The DDQN with prioritized experience replay algorithm uses a three-layer fully connected neural network to output Q values corresponding to all m action values. The choice of a simple three-layer fully connected neural network is driven by the need to minimize computational overhead, as DRLCAP dynamically adjusts GPU power. Furthermore, our environment is relatively straightforward, with only a five-dimensional input state and one-dimensional output action. Therefore, in such uncomplicated environments and tasks, overly complex neural network architectures can lead to overfitting or increased training difficulty.

The parameter θ^- of the target network Q^- is applied to evaluate the Q value of the optimal action, while the parameter θ of the current network Q is applied to choose the action corresponding to the largest Q value. To avoid overestimating the Q value, we separate the process for the action chosen and policy evaluation. Specifically, the target value is $r_j + \gamma Q^-\left(s_j\prime,\ argmax_{a\prime} Q\left(s_j\prime\ ,\ a\prime; \theta\right);\ \theta^-\right)$ (line 12 in Algorithm 1). The target network, with parameters θ^-, is identical to the online network except that its parameters are replicated from the online network every C steps. For each transition, the algorithm updates the network parameters (i.e., weights) using stochastic gradient ascent.

V. Experimental Setup

A. Platforms

We evaluate DRLCAP on three NVIDIA and one AMD GPU architectures, listed in Table I. Our main evaluation platform is an NVIDIA V100S GPU with Intel Xeon Silver 4210 CPU and 128 GB DDR4 memory. This GPU supports 196 frequency levels from 1597 MHz to 135 MHz, with a step of 7 MHz or 8 MHz. The power consumption of the entire GPU can be capped between 100 W and 250 W. To evaluate the portability of our approach, we also apply DRLCAP to NVIDIA RTX 3080 Ti GPU with Intel Xeon CPU E5-2680 v3 and 32 GB DDR4 memory, NVIDIA RTX 2080 Ti with Intel Xeon CPU E5-2697 v4 and 256 GB DDR4 memory, and AMD Radeon RX 6700 XT with Intel Xeon CPU E5-2698 v3 and 32 GB DDR4 memory. RTX 3080 Ti supports 126 core frequency levels and RTX 2080 Ti supports 120 core frequency levels.

TABLE I
LIST OF SPECIFICATIONS OF THE NVIDIA AND AMD GPUS.

Device	V100S	3080 Ti	2080 Ti	6700 XT
Architecture	Volta	Ampere	Turing	RDNA2
TDP (W)	250	350	257	211
Min Power Limit (W)	100	100	100	0
Max Power Limit (W)	250	380	280	211
Min Freq (MHz)	135	210	300	0
Max Freq (MHz)	1579	2130	2100	2855
Freq Step (MHz)	7/8	15	15	500,2855
Memory freq(MHz)	1107	405,810,5001, 9251,9501	405,810,5000, 6800,7000	96,456, 675,1000
Max Temp (C)	83	93	89	110

TABLE II
LIST OF BENCHMARKS USED IN THE TESTING PHASE.

Benchmark	Suite	Memory access	Category	Input
LULESH	LLNL	regular	memory	150
CloverLeaf	UK-MAC	irregular	memory	3840
TeaLeaf	UK-MAC	irregular	memory	4000
miniBUDE	UoB-HPC	regular	compute	65536
CoMD	NVIDIA	irregular	compute	100
Quantum ESPRESSO	Quantum ESPRESSO	irregular	memory	ausurf112
GROMACS	GROMACS	irregular	memory	water-0768
miniFE	Mantevo	regular	memory	400
VggNet-16	CNN	regular	compute	cifar-100
GoogleNet	CNN	regular	compute	cifar-100

We deploy the DRLCAP as a process on the CPU. The process uses nvidia-smi and Pynvml to adjust the NVIDIA GPU caps for the power and frequency respectively. For AMD GPU, the process uses rocm-smi to adjust the AMD GPU power and frequency cap. Note that NVIDIA RTX 2080 Ti and V100S do not support memory frequency capping and hence on these two GPUs, we do not perform memory frequency cappings. AMD RX 6700 XT only supports two core manual frequencies and four memory manual frequencies hence on this GPU, we replace perform frequency capping with power capping.

B. Training and Testing Benchmarks

During our evaluation, we train DRLCAP offline using training benchmarks and then test the trained model on new benchmarks that are not seen at the training stage. Specifically, DRLCAP is trained on five GPU benchmark suites: Rodinia [1], Parboil [2], SHOC [49], Polybench [50], and NVIDIA CUDA SDK. The trained model is then applied to ten representing benchmarks given in Table II. These benchmarks come from different suits, with distinct memory access patterns, program types, and input sizes. Our testing

TABLE III
LIST OF TWO PRIOR SCHEMES AND THREE IMPLEMENTATIONS OF DRLCAP.

Schemes	Describe	Dynamic
COORD	Heuristics based on program classification	No
RL	Based on Double Q-learning, frequency capping	Yes
Frequency	DRLCAP based on Double Deep Q-Network, frequency capping	Yes
Power	DRLCAP based on Double Deep Q-Network, power capping	Yes
Hybrid	DRLCAP based on Double Deep Q-Network, frequency and power capping	Yes

TABLE IV
LIST OF ACTION CANDIDATES ON V100S, 3080 TI, 2080 TI AND 6700 XT.

Configuration	Platform		Min	Max	Step
Power (W)	V100S		100	250	1
	6700 XT		60	211	3
Frequency (MHz)	V100S	Core	1035	1597	7/8
	3080 Ti	Core	1625	1925	15
		Memory	9501	9251	5001
	2080 Ti	Core	1635	2115	15

TABLE V
LIST OF HYPER-PARAMETERS OF DRLCAP.

Name	Value	Name	Value
Batch size	32	Replay buffer size	512
Learning rate (α)	0.001	Discount factor (γ)	0.95
Initial epsilon (ϵ)	0.3	Final epsilon (ϵ)	0.1
Training iteration	5000	Target network update frequency	8

benchmarks include LULESH [51], CloverLeaf, TeaLeaf [44], miniFE [52], CoMD [46], miniBUDE [45], two popular atomic and molecular applications (i.e., Quantum ESPRESSO (QE) and GROMACS [53]) and two CNN models (VggNet-16 and GoogleNet) training. All of these benchmarks are implemented in CUDA.

C. Competitive Schemes and DRL Configurations

We compare DRLCAP with four methods, as discussed in Section II. Table III provides a comprehensive list of these approaches. COORD is a static category-based heuristic for adjusting the GPU power. The RL-based approach employs the Double Q-learning to store two state-action tables to manage power. We also compare two variants of DRLCAP by using power capping and a combination of frequency and power capping (i.e., the hybrid scheme).

Table IV lists the power or frequency configurations considered by the DRL framework. In our implementation, we set the minimum value of the frequency cap to a higher value supported by the power management knobs since the lower capping can significantly negatively impact the application performance, as shown in Section III. Hyperparameter settings for DRLCAP are listed in Table V. All the code for reinforcement learning algorithms is written in Python 3.9 using TensorFlow 2.11. The code extends over 1000 lines.

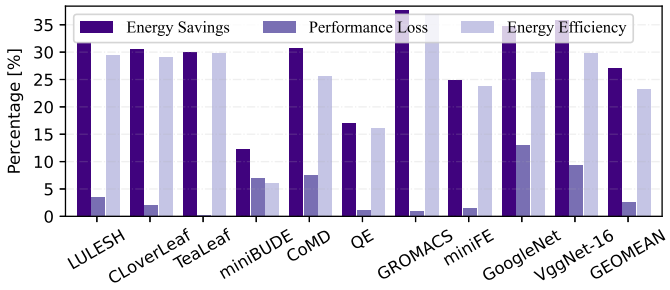

Fig. 9. Energy savings, the resulting slowdowns, and energy efficiency of DRLCAP with respect to the baseline scheme on V100S.

D. Performance Report

We report energy reduction and performance slowdown (loss) by comparing them to the default GPU power management strategy, where the GPU cap and memory cap run at the maximum frequency and are adjusted by the hardware according to the thermal constraint. We use the energy-delay product (EDP) to quantify the energy efficiency, computed as delay (seconds) $\times$ energy consumption. The EDP is a lower-is-better metric, and we show the reduction of EDP over the baseline. We run each test case 10 times on an unloaded machine. We then report the geometric mean across runs. We found the variance across runs is small, less than 1%.

VI. EXPERIMENTAL RESULTS

In this section, we first show that DRLCAP achieves consistently good performance across a broad spectrum of parallel applications on NVIDIA V100S. Then, we compare DRLCAP against the four alternative schemes described in Section V-C. Next, we evaluate the overhead of DRLCAP, including sampling overhead and computation and memory cost. Finally, we extend DRLCAP to two other NVIDIA GPU architectures and one AMD GPU to assess the portability.

A. Overall Results

Fig. 9 reports the energy saving and performance loss relative to the NVIDIA default scheme across benchmarks for DRLCAP on NVIDIA V100S. Our goal is to maximize energy saving (higher is better) with minimal performance loss (lower is better) for a high improvement of energy efficiency (measured by EDP - lower is better). Overall, DRLCAP achieves a geomean energy saving, performance loss and energy efficiency improvement are 27%, 2.4% and 23%, respectively, over the NVIDIA baseline.

For most test applications, DRLCAP reduces the GPU energy consumption at the cost of marginal performance losses. It can effectively improve energy efficiency for both computation-bounded and memory-bound programs. The only exception is miniBUDE, which has a low energy saving. This application is highly compute-bound which is very sensitive to the change of frequency capping as shown in Fig. 1. Further reducing the energy consumption with a lower frequency can lead to a high-performance loss for miniBUDE. For other computation-bounded applications, namely CoMD, VggNet-16 and GoogleNet, we can observe that the performance losses

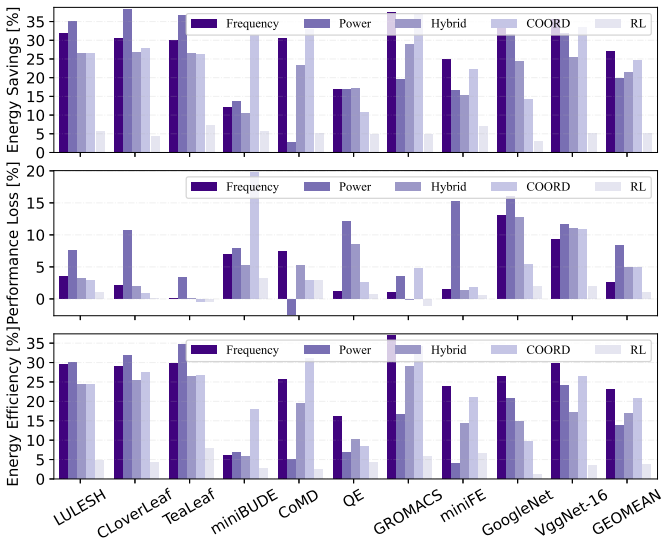

Fig. 10. Energy savings, the resulting slowdowns and EDP of five power management approaches with respect to the baseline scheme.

are higher than memory-bound applications since DRLCAP privatizes energy saving by reducing the GPU core frequency. For memory-bound applications, reducing the GPU core frequency has few impacts on performance. Additionally, due to DRLCAP being transparent to the applications, it saves energy for both regular and irregular applications.

B. Compare to Alternative Schemes

We now compare DRLCAP against the other four power management methods described in Table III. The results are given in Fig. 10, which shows that DRLCAP gives the best trade-off between energy saving and performance slowdown. The baseline is the NVIDIA default strategy.

COORD leads an average energy saving of 25% with an average performance slowdown of 5%, which translates to an improvement of energy efficiency of 21%. However, COORD needs to perform each program ahead of time to classify the program to assign a static power budget to be used through the program execution. Additionally, a static power strategy is inadequate for applications with frequent dynamic phase changes, such as Quantum ESPRESSO (QE). Compared to COORD, DRLCAP can adapt well to phase changes, leading to an 8% improvement in energy efficiency over COORD. However, in some cases, such as with CoDM, COORD outperforms our method. This is because COORD categorizes the CoDM as a memory-intensive application and configures its frequency to 1200 MHz. As shown in Fig. 2, the optimal frequency for the CoDM is 1185 MHz. Therefore, COORD has achieved the optimal configuration for this program. Additionally, COORD analyzes the program in advance without incurring any additional overhead during program execution, whereas our approach does incur some overhead.

The RL baseline gives an average energy saving of 5%, with an average performance slowdown of 1%, and an energy efficiency improvement of 4%. This is our previous approach targeting the multicore CPU. It can monitor hardware performance counters with low overhead on the CPU, like the Instruction Per Cycle (IPC). Thus, the scheme uses a simple RL algorithm, the double Q-learning, by using two tables to store all samples. However, this strategy fails as the number of states and actions increases because it is increasingly unlikely that the agent would visit a certain state or take a specific action. Therefore, although this scheme has adopted the same action candidates and reward function as DRLCAP, because of the large space of state and action, it fails to achieve similar energy conservation.DRLCAP leverages deep learning to progressively extract higher-level features from the raw state. In addition, two Q-tables occupy a large amount of memory space, which in reality does not make sense.

The mean energy savings of the DRLCAP variants when using power capping and the hybrid scheme are 20% and 22%, respectively. The corresponding mean performance losses are 8.5% and 5%, respectively. The mean energy efficiency improvements are 14% and 17%, respectively, over the NVIDIA baseline. Our chosen frequency capping scheme provides the best balance of energy savings and performance degradation for energy optimization. The main reason is that the NVIDIA V100S supports frequencies from 1597 MHz to 135 MHz, while the range of power limit is from 250 W to 100 W. Thus, the minimum frequency is well below the minimum power limit, DRLCAP with frequency capping can save more energy with lower hardware configurations. The power scheme shows significant energy savings for some benchmarks, such as LULESH, cloverleaf, Tealeaf, and miniBUDE. However, we also observed high performance losses for these programs, with LULESH, cloverleaf, and miniBUDE all exceeding 5%. The observation arises from the heightened sensitivity of these programs to fluctuations in power. As depicted in Fig. 2, TeaLeaf and miniBUDE show significant fluctuations in both energy consumption and performance as power varies, compared to QE and COMD. Conversely, as illustrated in Fig. 1, all programs exhibit consistent sensitivity to frequency changes. The optimization of the hybrid scheme is slightly worse than that of the frequency capping scheme because of the interaction of the power and frequency capping and the large hybrid action space. In real-world scenarios, the relationship between GPU power consumption and frequency is typically nonlinear as shown in Fig. 3. This means that reducing the frequency does not result in a linear reduction in power consumption. The hybrid scheme can sometimes lead to a scenario in which the frequency decreases, but the power consumption remains relatively high. This situation can hinder the achievement of optimal energy efficiency. Additionally, the hybrid scheme may occasionally result in overly conservative adjustments, which in turn restrict the GPU performance potential.

C. Overhead Analysis

The overhead of DRLCAP when applied comes from two sources: the cost of online sampling for workload change detection, and the cost of running the system for computation and memory. Additionally, we explore the variations in overhead introduced by different neural network architectures.

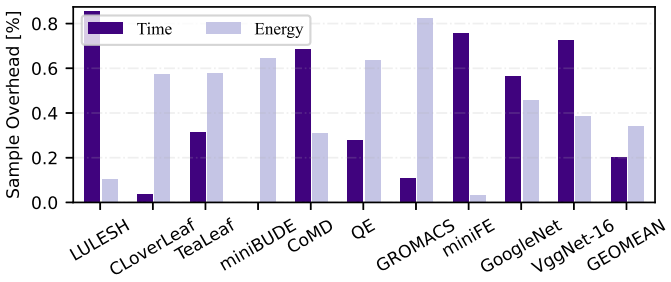

Fig. 11. Energy and time overhead with respect to the baseline scheme without sampling on V100S.

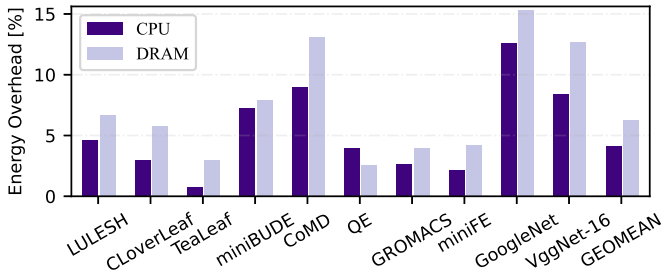

Fig. 12. Energy overhead of CPU and memory with respect to the baseline scheme.

Firstly, we evaluate the overhead of our sampling approach which is determined by the vendor software nvidia-smi. Fig. 11 illustrates the sampling overhead of time and energy provided by the interval of 300 ms to the baseline that does not incur sampling overhead. As can be seen from the diagram, the geometric means of time overhead and energy overhead are 0.2% and 0.3%.In this work, we choose 300 ms as the sampling interval that has a negligible negative influence on energy and time. After the sampling is completed, DRLCAP updates the GPU frequency cap if the best action has changed. If the sampling window is too large, the phase change detection may not be timely, and too small may cause the calculation of DRLCAP to be too dense and the overhead to be large.

Another overhead depends on the complexity of our DRL-based GPU energy efficiency optimization algorithm. Since DRLCAP is deployed in the host (CPU), we now evaluate the CPU and memory energy overhead. Fig. 12 illustrates the CPU and memory energy overhead with respect to the case without sampling. The NVIDIA V100S is fitted with one 40-core Intel Xeon Silver 4210 CPU and 128G DDR4. As can be seen from the diagram, the geomean CPU and memory energy overhead with DRLCAP are 4% and 6%. Since the power consumption is low when the CPU is unloaded, slight CPU activity will also result in a higher percentage of power consumption. We also evaluate the energy consumption of the overall system including CPU, memory and GPU. Fig. 13 shows the energy saving and energy efficiency improvement of the overall computing system. DRLCAP with frequency causes energy overhead, while it still achieves an average overall energy saving of 17%, a performance loss of 2.4% and an improvement of energy efficiency of 12%. The DRL algorithm results in a certain amount of energy overhead from CPU and memory, however, DRLCap achieves energy efficiency optimization across the entire computing system.

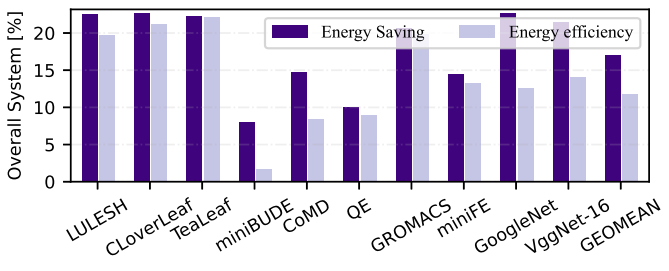

Fig. 13. Overall energy saving and energy efficiency with DRLCAP.

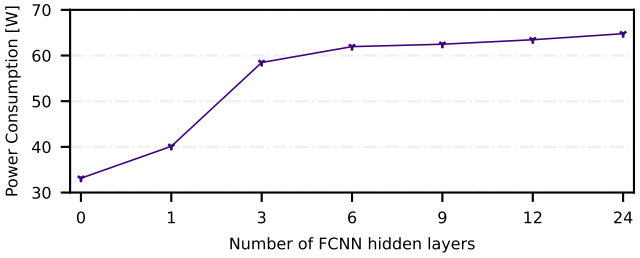

Fig. 14. Power consumption with different number of hidden layers.

In this study, we utilize a relatively simple FCNN structure, specifically a three-layer fully connected neural network. This choice is made due to the dynamicity of DRLCAP, necessitating consideration of system overhead at runtime. Fig. 14 depicts the power consumption of the computer system both without and with DRLCAP, equipped with various neural networks. The X-axis represents the hidden layer of the neural network, with 0 denoting the case without DRLCAP, while the Y-axis indicates the average power consumption over 10 measurements for 2 minutes. We can observe that the power consumption increases as the number of neural network layers increases. For instance, with a single hidden layer, power consumption reaches 40 W, compared to 33 W without DRLCAP. With three hidden layers, power consumption rises to 58 W, which is deemed intolerable. Consequently, for this study, we selected a neural network with a simple structure, featuring only one hidden layer.

D. Portability

So far, all our experiments have been performed on NVIDIA V100S GPU. In this evaluation, we also applied our approach to the NVIDIA GeForce RTX 3080 Ti, NVIDIA GeForce RTX 2080 Ti and AMD Radeon RX 6700 XT GPUs(Section V-A). We need to re-train the DRL model when moving to a new architectural GPU. Figs. 15 and 16 show the result on the three NVIDIA GPUs and one AMD GPU targeted in this paper respectively. Once again, the results are normalized to the default baseline.

For the NVIDIA Ampere architecture (RTX 3080 Ti), the average energy reduction, performance slowdown and energy efficiency improvement are 19%, 3% and 17%, respectively. We also observe similar results on the NVIDIA Tuning architecture (RTX 2080 Ti). The average energy reduction, performance loss and improvement of energy efficiency are 22%, 4.5% and 19% on RTX 2080 Ti. Across the three GPU architectures, DRLCAP saves the energy consumption, on

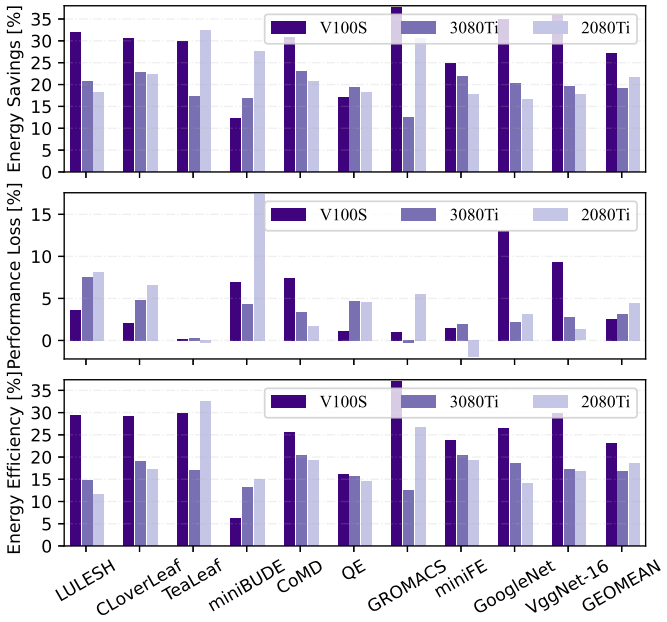

Fig. 15. Energy savings, the resulting slowdowns and EDP with DRLCAP on three NVIDIA GPUs

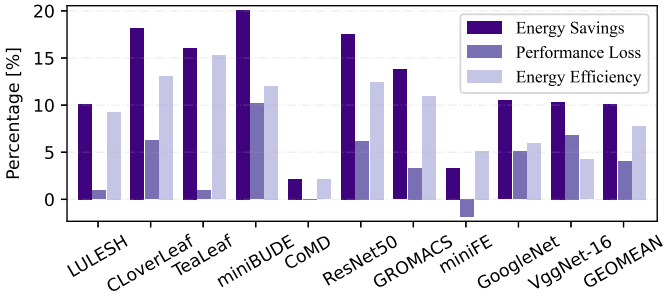

Fig. 16. Energy savings, the resulting slowdowns and EDP with DRLCAP on AMD Radeon RX 6700 XT GPU.

average, by 22%, reduces performance by 3%, and improves energy efficiency by 20%. For the AMD RDNA 2 architecture (RX 6700 XT), the average energy saving, performance loss and energy efficiency improvement are 10%, 4% and 7.8%, respectively. Since AMD GPUs only support manual tuning at two frequency levels, we choose to adjust the power capping. As a result, we achieve fewer energy savings on AMD than we do on NVIDIA GPUs. The GPU version of Quantum ESPRESSO (QE) requires the nvfortran compiler from the NVIDIA HPC SDK. Therefore, instead of QE, we use ResNet50, which is also a convolutional neural network like GoogleNet and VggNet. We also observe that energy savings are poor at CoMD and miniFE. The energy saving potential of these two benchmarks is very low as we statically traverse all the power caps and the achieved energy savings are very low. In general, DRLCAP exhibits good portable performance across the evaluation platforms.

VII. Discussion and Future Work

DRLCap is among the first work that tries to apply deep reinforcement learning to optimize GPU energy consumption by frequency capping during dynamic runtime. Nevertheless, there is still space for development and more work should be done.

Model interpretability. Machine learning generally works as a black box. This is just as true for the DRL technique used in this work. Three distinct challenges of explainable reinforcement learning, namely, environmental interpretation, task interpretation, and strategy interpretation, are defined according to the characteristics of RL. The environment is a given black-box system with specific internal rules, and how to obtain these rules from the environment is the first problem facing reinforcement learning. A task is an objective function fitted by an agent to maximize its average reward. However, the reward function based on prior human knowledge is usually subjective and sparse and cannot accurately represent task goals. The strategy interpretation needs to be presented in an interpretable manner so that the human predictions of the model's actions are as close as possible to the model's actual actions. It is an interesting future work direction to understand the DRL learning mechanism.

GPU Performance metrics. One of the challenges for GPU energy efficiency optimization is how to accurately and low-overhead measure the runtime performance. There are two major approaches in the existing research. The first is based on hardware performance counters [54]. This method measures hardware performance count through the performance profiling tools. However, until the GPU kernel terminates, these profiling tools return measurements, which means that the approach is offline technology because the GPU kernel must be executed at least once. In addition, the overhead of obtaining these indicators is usually high. The second approach is based on code instrumentation, such as CUDAAdvisor [55], to measure statistics about the control flow. Although the code instrumentation has little performance effect than the former, it changes the actual behavior of GPU kernels and requires extra effort made by developers and the system administrator in code changes and maintenance. In this work, DRLCAP utilizes the hardware utilization provided by nvidia-smi based on NVML as performance metrics. Although it avoids the disadvantages of the two popular methods mentioned above, it is coarse-grained and fails to accurately reflect the performance of the program. As a result, it is an interesting future work to capture accurate and low-overhead performance metrics.

Global optimization. DRLCAP can allocate one agent to each GPU, and the deployed agent takes choices independently on the local GPU. Our solution can be scalable to a large, distributed environment thanks to this decentralized design. To increase cluster throughput and energy efficiency, certain lightweight approaches to coordinate executions and optimizations among distributed computing nodes would be interesting to develop. For example, we can leverage GEOPM's tree-hierarchical optimization framework [10] to propagate data and leverage feedback from each computing node to coordinate power optimization. Particularly, DRLCAP with power capping is useful for a power constraint distributed environment since the feedback in our scheme may be used to distribute the power budget in a distributed environment under an overall power limit.

Frequency and Voltage. Most current DVFS methods focus on the frequency while disregarding the voltage, which limits their ability to fully maximize energy savings. DVFS just lowers the voltage to comply with frequency reduction in the slack from load imbalance, communication delay, memory access latency, etc. The neglected undervolting technology is advantageous in several ways: (a) it allows for a decrease in the voltage supplied to the component without changing its frequency which means that the computational throughput can be maintained at a similar level as before, ensuring that the performance of the system is not significantly affected. (b) it can be universally used for both slack and non-slack phases of HPC runs. By reducing the voltage during these phases, undervolting helps in achieving power reduction across the entire workload, regardless of the level of computational intensity. Hence, in future research, we can combine undervolting and frequency capping to further improve GPU energy efficiency. Note that undervolting can lead to an increased system failure rate. Consequently, we need to implement various resilience techniques to mitigate and tolerate these failures in future work.

VIII. Conclusion

We have presented DRLCap, a general deep reinforcement learning-based online GPU power management scheme cross various GPU architectures. DRLCap aims to reduce the GPU energy consumption without significantly compromising the application runtime by adjusting the GPU frequency caps at program execution time, which is achieved by the RL reward. DRLCap adapts to dynamic workload behaviors and updates its decision process during execution time by considering both immediate and prospective future feedback of power optimization, aiming to maximize the long-term reward throughout the program execution. We evaluate DRLCap by applying it to three NVIDIA and one AMD GPU architectures using ten GPU benchmarks. Experimental results show that DRLCap can reduce the GPU energy consumption by 22%, on average, at the cost of 3% slowdown of the application runtime on NVIDIA GPUs. For the AMD GPU, DRLCap saves energy consumption by 10%, on average, with 4% performance loss.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (Grant No. 62202123) and in part by the Joint Funds of the National Natural Science Foundation of China (Grant No. U22A2036). Prof. Zhang is the corresponding author.

References

[1] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. 2009 IEEE Int. Symp. Workload Charact.* IEEE, 2009, pp. 44–54.

[2] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Cent. Reliable High Perform. Comput.*, vol. 127, p. 27, 2012.

[3] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 873–880.

[4] T. Patki, N. Bates, G. Ghatikar, A. Clausen, S. Klingert, G. Abdulla, and M. Sheikhalishahi, "Supercomputing centers and electricity service providers: A geographically distrib. perspective on demand management in europe and the united states," in *Int. Conf. High Perform. Comput.* Springer, 2016, pp. 243–260.

[5] T. Patki, Z. Frye, H. Bhatia, F. Di Natale, J. Glosli, H. Ingolfsson, and B. Rountree, "Comparing gpu power and frequency capping: A case study with the mummi workflow," in *Proc. 2019 IEEE/ACM Workflows Support Large-Scale Sci.* IEEE, 2019, pp. 31–39.

[6] Q. Zhu, B. Wu, X. Shen, L. Shen, and Z. Wang, "Co-run scheduling with power cap on integrated cpu-gpu systems," in *Proc. 31st IEEE Int. Parallel Distrib. Process. Symp.* IEEE, 2017, pp. 967–977.

[7] Y. Wang, W. Zhang, M. Hao, and Z. Wang, "Online power management for multi-cores: A reinforcement learning based approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 4, pp. 751–764, 2022.

[8] M. Hao, W. Zhang, Y. Wang, G. Lu, F. Wang, and A. V. Vasilakos, "Fine-grained powercap allocation for power-constrained systems based on multi-objective machine learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1789–1801, 2020.

[9] L. Chen, P. Wu, Z. Chen, R. Ge, and Z. Zong, "Energy efficient parallel matrix-matrix multiplication for dvfs-enabled clusters," in *2012 41st International Conference on Parallel Processing Workshops*. IEEE, 2012, pp. 239–245.

[10] J. Eastep, S. Sylvester, C. Cantalupo, B. Geltz, F. Ardanaz, A. Al-Rawi, K. Livingston, F. Keceli, M. Maiterth, and S. Jana, "Global extensible open power manager: a vehicle for hpc community collaboration on co-designed energy management solutions," in *Proc. 32nd Int. Conf. High Perform. Comput.* Springer, 2017, pp. 394–412.

[11] C. Ortega, L. Alvarez, M. Casas, R. Bertran, A. Buyuktosunoglu, A. E. Eichenberger, P. Bose, and M. Moreto, "Intelligent adaptation of hardware knobs for improving performance and power consumption," *IEEE Trans. Comput.*, vol. 70, no. 1, pp. 1–16, 2020.

[12] Y. Luo, L. Pu, and C.-H. Liu, "Cpu frequency scaling optimization in sustainable edge computing," *IEEE Transactions on Sustainable Computing*, 2022.

[13] L. Tan, Z. Chen, Z. Zong, D. Li, and R. Ge, "A2e: Adaptively aggressive energy efficient dvfs scheduling for data intensive applications," in *2013 IEEE 32nd International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2013, pp. 1–10.

[14] B. Salami, H. Noori, and M. Naghibzadeh, "Fairness-aware energy efficient scheduling on heterogeneous multi-core processors," *IEEE Trans. Comput.*, vol. 70, no. 1, pp. 72–82, 2020.

[15] S. Kumar, A. Gupta, V. Kumar, and S. Bhalachandra, "Cuttlefish: library for achieving energy efficiency in multicore parallel programs," in *Proc. Int. Conf. High Perform. Comput., Networking, Storage Anal.*, 2021, pp. 1–14.

[16] J. Leng, A. Buyuktosunoglu, R. Bertran, P. Bose, and V. J. Reddi, "Safe limits on voltage reduction efficiency in gpus: a direct measurement approach," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 294–307.

[17] H. Zamani, Y. Liu, D. Tripathy, L. Bhuyan, and Z. Chen, "Greenmm: energy efficient gpu matrix multiplication through undervolting," in *Proceedings of the ACM International Conference on Supercomputing*, 2019, pp. 308–318.

[18] H. Zamani, D. Tripathy, L. Bhuyan, and Z. Chen, "Saou: safe adaptive overclocking and undervolting for energy-efficient gpu computing," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 205–210.

[19] L. Tan, S. L. Song, P. Wu, Z. Chen, R. Ge, and D. J. Kerbyson, "Investigating the interplay between energy efficiency and resilience in high performance computing," in *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2015, pp. 786–796.

[20] J. Guerreiro, A. Ilic, N. Roma, and P. Tomas, "Gpgpu power modeling for multi-domain voltage-frequency scaling," in *Proc. 24th IEEE Int. Symp. High Perform. Comput. Archit.* IEEE, 2018, pp. 789–800.

[21] L. Braun, S. Nikas, C. Song, V. Heuveline, and H. Fröning, "A simple model for portable and fast prediction of execution time and power consumption of gpu kernels," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 1, pp. 1–25, 2020.

[22] P. Zou, A. Li, K. Barker, and R. Ge, "Indicator-directed dynamic power management for iterative workloads on gpu-accelerated systems," in *Proc. 20th IEEE/ACM Int. Symp. Cluster, Cloud Internet Comput.* IEEE, 2020, pp. 559–568.

[23] F. Wang, W. Zhang, S. Lai, M. Hao, and Z. Wang, "Dynamic gpu energy optimation for machine learning training workloads," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2943–2954, 2022.

[24] J.-G. Park, N. Dutt, and S.-S. Lim, "An interpretable machine learning model enhanced integrated cpu-gpu dvfs governor," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 6, pp. 1–28, 2021.

[25] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Performance and power prediction for concurrent execution on gpus," *ACM Trans. Archit. Code Optim.*, vol. 19, no. 3, pp. 1–27, 2022.

[26] R. Ge, X. Feng, T. Allen, and P. Zou, "The case for cross-component power coordination on power bounded system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 10, pp. 2464–2476, 2021.

[27] H. Hasselt, "Double q-learning," *Adv. Neural Inf. Process. Syst.*, vol. 23, 2010.

[28] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. 4th Int. Conf. Learn. Represent.*, 2015.

[29] Y. Arafa, A.-H. Badawy, A. ElWazir, A. Barai, A. Eker, G. Chennupati, N. Santhi, and S. Eidenbenz, "Hybrid, scalable, trace-driven performance modeling of gpgpus," in *Proc. Int. Conf. High Perform. Comput., Networking, Storage Anal.*, 2021, pp. 1–15.

[30] V. Kandiah, S. Peverelle, M. Khairy, J. Pan, A. Manjunath, T. G. Rogers, T. M. Aamodt, and N. Hardavellas, "Accelwattch: A power modeling framework for modern gpus," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2021, pp. 738–753.

[31] A. Li, S. L. Song, M. Wijtvliet, A. Kumar, and H. Corporaal, "Sfu-driven transparent approximation acceleration on gpus," in *Proc. 2016 Int. Conf. Supercomput.*, 2016, pp. 1–14.

[32] A. Majumdar, L. Piga, I. Paul, J. L. Greathouse, W. Huang, and D. H. Albonesi, "Dynamic gpgpu power management using adaptive model predictive control," in *Proc. 23rd IEEE Int. Symp. High Perform. Comput. Archit.* IEEE, 2017, pp. 613–624.

[33] I. Paul, W. Huang, M. Arora, and S. Yalamanchili, "Harmonia: Balancing compute and memory power in high-performance gpus," *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, vol. 43, no. 3S, pp. 54–65, 2015.

[34] S. Z. Sheikh and M. A. Pasha, "Energy-efficient cache-aware scheduling on heterogeneous multicore systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 206–217, 2022.

[35] Q. Wang, X. Mei, H. Liu, Y.-W. Leung, Z. Li, and X. Chu, "Energy-aware non-preemptive task scheduling with deadline constraint in dvfs-enabled heterogeneous clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4083–4099, 2022.

[36] L. Gu, D. Zeng, W. Li, S. Guo, A. Y. Zomaya, and H. Jin, "Intelligent vnf orchestration and flow scheduling via model-assisted deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 279–291, 2019.

[37] D. Zeng, J. Zhang, L. Gu, S. Guo, and J. Luo, "Energy-efficient coordinated multipoint scheduling in green cloud radio access network," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 10, pp. 9922–9930, 2018.

[38] L. Gu, D. Zeng, S. Guo, and B. Ye, "Leverage parking cars in a two-tier data center," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2013, pp. 4665–4670.

[39] L. Gu, D. Zeng, S. Tao, S. Guo, H. Jin, A. Y. Zomaya, and W. Zhuang, "Fairness-aware dynamic rate control and flow scheduling for network utility maximization in network service chain," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1059–1071, 2019.

[40] Z. Tian, Z. Wang, J. Xu, H. Li, P. Yang, and R. K. V. Maeda, "Collaborative power management through knowledge sharing among multiple devices," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 38, no. 7, pp. 1203–1215, 2018.

[41] H. Huang, M. Lin, L. T. Yang, and Q. Zhang, "Autonomous power management with double-q reinforcement learning method," *IEEE Trans. Ind. Inform.*, vol. 16, no. 3, pp. 1938–1946, 2019.

[42] R. Azimi, C. Jing, and S. Reda, "Powercoord: Power capping coordination for multi-cpu/gpu servers using reinforcement learning," *Sustainable Comput.: Inf. Syst.*, vol. 28, p. 100412, 2020.

[43] E. Kwon, S. Han, Y. Park, J. Yoon, and S. Kang, "Reinforcement learning-based power management policy for mobile device syst." *IEEE Trans. Circuits Syst.*, vol. 68, no. 10, pp. 4156–4169, 2021.

[44] "Uk mini-app consortium." https://uk-mac.github.io/.

[45] A. Poenaru, W.-C. Lin, and S. McIntosh-Smith, "A performance analysis of modern parallel programming models using a compute-bound application," in *Proc. 36th Int. Conf. High Perform. Comput.* Springer, 2021, pp. 332–350.

[46] J. Mohd-Yusof, S. Swaminarayan, and T. C. Germann, "Co-design for molecular dynamics: An exascale proxy application," *LA-UR 13-20839*, pp. 88–89, 2013.

[47] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo *et al.*, "Quantum espresso: a modular and open-source software project for quantum simulations of materials," *J. Phys.: Condens. Matter*, vol. 21, no. 39, p. 395502, 2009.

[48] V. Spiliopoulos, S. Kaxiras, and G. Keramidas, "Green governors: A framework for continuously adaptive DVFS," in *2011 Int. Green Comput. Conf. Workshops*. IEEE Comput. Society, 2011, pp. 1–8.

[49] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (shoc) benchmark suite," in *Proc. 3rd Workshop General-purpose Comput. Graphics Process. Units*, 2010, pp. 63–74.

[50] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, and J. Cavazos, "Auto-tuning a high-level language targeted to gpu codes," in *Proc. 2012 Innovative Parallel Comput.* Ieee, 2012, pp. 1–10.

[51] I. Karlin, J. Keasler, and J. R. Neely, "Lulesh 2.0 updates and changes," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2013.

[52] "Ecp proxy applications." https://proxyapps.exascaleproject.org/.

[53] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, "Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomput.s," *SoftwareX*, vol. 1-2, pp. 19–25, 2015.

[54] Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, and M. Peres, "Power and performance characterization and modeling of gpu-accelerated systems," in *2014 IEEE 28th Int. Parallel Distrib. Process. Symp.* IEEE, 2014, pp. 113–122.

[55] D. Shen, S. L. Song, A. Li, and X. Liu, "Cudaadvisor: Llvm-based runtime profiling for modern gpus," in *Proc. 2018 Int. Symp. Code Gener. Optim.*, 2018, pp. 214–227.

Yiming Wang received the MS degree in software engineering from Harbin Institute of Technology, China, in 2019. She is currently working toward a Ph.D. degree in Harbin Institute of Technology. Her research interests include high-performance computing, per-formance optimization, and energy efficiency.

Meng Hao received the BS and Ph.D. degree in computer science and engineering from Harbin Institute of Technology, China, in 2014 and 2020 respectively. He is currently an assistant professor in Harbin Institute of Technology. His research interests include high-performance computing, performance modeling, and parallel optimization.

Hui He received the B.Eng, M.Eng, and Ph.D. degrees from the Harbin Institute of Technology, Harbin, China, in 1997, 1999, and 2006, respectively, all in computer science and technology. She is a Professor in Harbin Institute of Technology. Her current research interests include distributed computing, the Internet of Things, and big data analysis.

Weizhe Zhang (Senior Member, IEEE) received B.Eng, M.Eng and Ph.D. degree of Engineering in computer science and technology in 1999, 2001 and 2006 respectively from Harbin Institute of Technology. He is currently a professor at Harbin Institute of Technol-ogy, China. His research interests are primarily in parallel computing, distributed computing, cloud and grid computing, and computer network.

Qiuyuan Tang received the BS degree in computer science and engineering from the Harbin Institute of Technology, China, in 2022. She is currently working in Shanghai Bili Bili Technology Co., Ltd.

Xiaoyang Sun is a Ph.D. student of the Distributed Systems and Services Group at the University of Leeds. He has completed research internships in Alibaba Group, working on efficient resource management in the data center and deep learning acceleration in a resource-restricted environment. His primary research interests include edge computing, system optimization for deep learning, etc.

Zheng Wang is a professor of intelligent software technology at the University of Leeds. His research interests include compilers, programming models, parallel computing, runtime systems, and systems security.