Online Power Management for Multi-cores: A Reinforcement Learning Based Approach

Yiming Wang, Weizhe Zhang, *Senior Member, IEEE*, Meng Hao, Zheng Wang, *Member, IEEE*

Abstract—Power and energy is the first-class design constraint for multi-core processors and is a limiting factor for future-generation supercomputers. While modern processor design provides a wide range of mechanisms for power and energy optimization, it remains unclear how software can make the best use of them. This paper presents a novel approach for runtime power optimization on modern multi-core systems. Our policy combines power capping and uncore frequency scaling to match the hardware power profile to the dynamically changing program behavior at runtime. We achieve this by employing reinforcement learning (RL) to automatically explore the energy-performance optimization space from training programs, learning the subtle relationships between the hardware power profile, the program characteristics, power consumption and program running times. Our RL framework then uses the learned knowledge to adapt the chip's power budget and uncore frequency to match the changing program phases for any new, previously unseen program. We evaluate our approach on two computing clusters by applying our techniques to 11 parallel programs that were not seen by our RL framework at the training stage. Experimental results show that our approach can reduce the system-level energy consumption by 12%, on average, with less than 3% of slowdown on the application performance. By lowering the uncore frequency to leave more energy budget to allow the processor cores to run at a higher frequency, our approach can reduce the energy consumption by up to 17% while improving the application performance by 5% for specific workloads.

Index Terms—power management, multi-cores, reinforcement learning, power capping, uncore frequency, phase change detection.

1 Introduction

In an era where computing hardware hits the power wall, energy efficiency is of paramount importance to today's computing systems. Indeed, power and energy consumption is the first-class constrain for high-performance computing (HPC) systems and future generation exascale computing infrastructures [1], [2]. To improve the energy efficiency of next-generation exascale computing, we need to significantly improve the energy efficiency of computer systems to make HPC scalable and sustainable.

Modern processors provide a range of techniques for energy optimization. Examples of such techniques include Dynamic Voltage and Frequency Scaling (DVFS) [3], Intel's Running Average Power Limit (RAPL) [4] and AMD's Thermal Design Power (TDP) Power Cap [5]. These mechanisms offer energy-saving potential by allowing the software to monitor and control the power profile of processor cores and their peripherals. Realizing such potential requires the software system to dynamically reconfigure the hardware to match the behavior and demand of the running application. However, doing so is challenging as the best hardware configuration changes from one program to the other and from one running phase to another within a single program execution.

There is an extensive body of work on utilizing the hardware power control mechanism to reduce energy consumption of computing systems [6], [7], [8], [9], [10]. While encouraging results have been achieved, existing solutions primarily focus on optimizing the energy consumption of the CPU core. They largely overlook other CPU subsystems for memory communications, cache coherence, and input and output peripherals, which are referred to as the *uncore* by Intel. Power optimization for the uncore subsystem is increasingly crucial because it can account for 20% of the overall CPU power consumption [11], [12], [13], and this contribution is expected to grow for future generation CPUs [14], [15], [16]. Uncore power optimization is also particularly important for emerging HPC workloads like large-scale data processing applications, which often incur extensive data communications [17], [18]. As we will show later in the paper, leaving the hardware to manage the uncore frequency often results in a significant waste of energy consumption that a more intelligent software-based optimization scheme could otherwise save.

This paper presents a new approach for optimizing multi-core power consumption by dynamically matching the CPU configuration (i.e., the maximum power limit of the multi-core chip and its uncore frequency) to the running program. As a departure from prior works [19], [20], [21], our approach explicitly considers the frequency of the uncore subsystem for energy optimization. We achieve this by developing an adaptive power management scheme to limit, *at runtime*, the multi-core chip's power consumption (e.g., power cap)[1] and configure the uncore frequency. Our approach dynamically adjusts the power profile of the CPU to match the workload states (or phases) throughout its execution. Such an online adaption approach allows the power

- *Y. Wang, W. Zhang and M. Hao are with the School of Cyberspace Science, Harbin Institute of Technology, China. E-mail: {yimingw, wzzhang, haomeng}@hit.edu.cn*
- *Z. Wang is with the School of Computing, University of Leeds. E-mail: z.wang5@leeds.ac.uk*

1. DVFS of the CPU is being moved to hardware on modern multi-cores like Intel Xeon processors [22]. However, we can still configure the power cap of the CPU to guide hardware DVFS.

manager to dynamically tailor the hardware configuration to the changing program execution characteristics. It avoids the pitfalls of a static optimization strategy where the hardware configuration remains unchanged for the dynamically evolving program phases [23], [24], [25], [26].

One of the key challenges of online power management is how to detect phase changes and adapt to such changes. The state-of-the-art machine-learning-based power management method [10] uses the slack between the idle time and the wall time of the *entire CPU* to detect phase changes. However, such a strategy cannot decouple the behaviors between the core and the uncore domains. As we will show later in Section 5.2, this approach is inadequate for uncore power optimization, leaving much room for improvement. By contrast, our work leverages the hardware performance counters to decouple core and uncore domains, providing a more accurate mechanism for detecting uncore phase changes. To adapt to the changing runtime behavior, we then dynamically re-configure the hardware when a program phase change is detected.

Unlike prior online power optimization methods that focus on choosing a short-term power configuration for the current observation (that can be sub-optimal for the longer term) [7], [27], [28], [29], our approach is designed to optimize the overall power optimization for the entire program execution. Here, the central issue is: how do we, at runtime, evaluate whether a particular configuration is good? We cannot afford to try out all configurations and pick the best. Furthermore, once we have selected a policy and followed its decision, we still do not know how good it was in the longer term.

We overcome these challenges by employing reinforcement learning (RL) [30] to quickly explore the optimization space to learn how to apply a power configuration based on the current workload state and adapt its decision during program execution. Specifically, we learn a policy network to determine what CPU configuration to apply given the current system state. The policy network aims to maximize the cumulative reward, i.e., the overall energy reduction to the performance loss, of the entire program execution period. To train the network, we utilize the recently proposed double Q-learning [31] algorithm, which is shown to be effective in learning over a large, complex optimization space and can avoid local optimal due to the overestimation of rewards. During training, the RL agent refines and adjusts its prediction based on the measured power consumption, so that a more appropriate decision can be made for the next scheduling epoch. The agent first learns what action to use for a given system state from training programs. The learned agent can then be applied to any new, unseen program during deployment.

Unlike prior machine-learning-based approaches [23], [24], [25], [26], [32], where the learned model remains static after deployment, we use RL to continually refine and update the decision policy throughout runtime execution. As the RL system learns and adjusts its decisions over time, it gains a better understanding of what works for the running program and becomes more efficient in recommending hardware power configurations for the target programs and underlying hardware. Our approach is decentralized, where we deploy an RL agent to each computing node to monitor the phase changes and act accordingly on the local node. This allows us to deal with situations where processes on different nodes do not synchronize perfectly in phases.

Compared to supervised-learning methods [23], [24], [25], [33], [34], [35], [36], [37], [38], our RL-based solution has the benefit of not requiring labelled a large number of training samples to train the model. Obtaining sufficient and representative training samples to cover a diverse set of workloads seen in deployment have been shown to be difficult [39], [40], [41], [42]. Without a large and sufficient training dataset, a supervised learning method often delivers poor performance during real-life uses as the target program behavior can be significantly different from those seen at the training phase. Our work avoids this pitfall by first using offline learning to boost the learning of a decision agent and then use runtime feedback to update the learned knowledge constantly. Our approach is useful for the typical long-running workloads in an HPC environment. It allows the power management system to adapt to the dynamic program behavior that can change depending on the program input, which is hard to anticipate ahead of time.

We evaluate our approach by applying it to 19 parallel benchmarks running on two HPC clusters, including a 4-node cluster with 160 Cascade Lake cores and a 16-node cluster with 196 Haswell cores. We compare our approach against three state-of-the-art multi-core power management systems [10], [43], [44], and two implementation variants of our RL-based approach. Experimental results show that our scheme consistently outperforms prior methods, reducing the energy consumption and program running time respectively by 12% and 3% on average. We show that our approach achieves this without significantly compromising the program execution time compared to a baseline scheme for setting the power cap to the TDP and letting the Linux `ondemand` frequency governor dynamically determine the core frequency and the CPU firmware to modulate the uncore frequency. Experimental results show that our system exhibits a good scalability by delivering comparable performance on the two clusters with different computing nodes and sizes. We show that, in some cases, we can accelerate the program runtime by 3% while reducing the energy consumption by 17%. Our work focuses on modern multi-core CPUs, a fundamental building block of exascale computing. It targets power and energy consumption, a limiting factor of exascale computing. Moreover, our decentralized decision process has good scalability and can be extended to a large distributed environment.

This paper makes the following contributions:

- It is the first work to employ RL to dynamically configure the uncore frequency for power management, by simultaneously considering the chip power capping and the frequency of the uncore domain.
- It presents a simple yet effective approach for automatic and dynamic phase detection that decouples core and uncore domains. Our approach does not require user involvement and is transparent to the running application.
- It provides a detailed analysis of the working mechanism of RL-based online power management on real computing clusters.

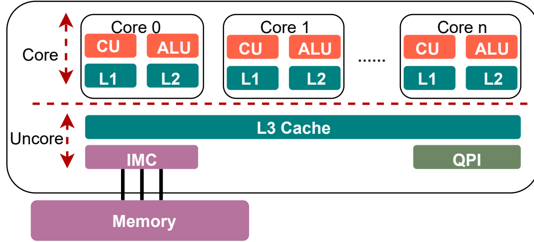
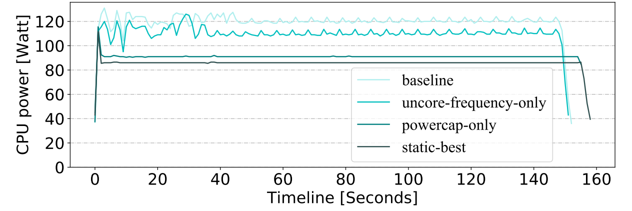
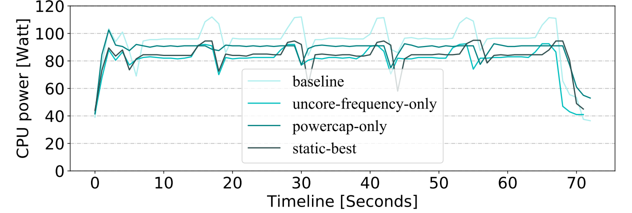


Fig. 1: A simplified view of processor cores and the uncore subsystem of the Intel processor design.



(a) AMG.



(b) miniQMC.

Fig. 2: CPU power consumption and application runtime for benchmarks AMG (a) and miniQMC (b).

2 Background and Motivation

2.1 Uncore and Power Management Interfaces

Our work targets the Intel processor architecture, but the methodology can be applied to other multi-core architectures with power monitoring and configuration interfaces. For instance, our techniques can be easily ported to the AMD Zen architecture that also provides RAPL-like interfaces for power monitoring and configurations [45], [46].

Figure 1 depicts a simplified view of an Intel processor that contains both processor cores and the uncore. The core contains the components of the processor involved in executing instructions, including the arithmetic logic unit (ALU), floating-point unit (FPU), and the Level 1 and Level 2 caches. The uncore functions include the quick path interconnect (QPI) controller, the integrated memory controllers (IMC), and the last level cache (LLC). The uncore typically occupies 30% of a die area [14] and can contribute to 20% of the processor's power consumption [12], [13]. We note that the current Intel CPU firmware sets the uncore to run at the highest frequency throughout the program execution once an uncore activity is detected (e.g., a memory load). Such a strategy can lead to significant energy waste for CPU-bound applications where the application performance is insensitive to the memory latency. Our work is designed to avoid this drawback by dynamically adjusting the uncore frequency.

Power monitoring and control. In this work, we limit the processor's power consumption by configuring some RAPL-related, model-specific registers (MSR) on our evaluation platforms. Specifically, we use the open-source `msr-safe` Linux kernel module [47] to read from and write to the MSRs for power measurement and uncore frequency scaling (UFS). The `msr-safe` module provides user-level interfaces that do not require root permission. We also use the Linux `perf` profiler to read other performance counters for phase change detection (Section 3.2). For the events that we trace through `perf`, no root privilege is required. Finally, to set the power cap, we use the Linux `powercap` interface that also does not need root permission.

2.2 Reinforcement Learning

Our work employs RL to develop a dynamic power optimization scheme. RL works by obtaining strategy improvement through continuous interactions with the changing environment in discrete time steps [48]. At each step, an agent receives the current state and reward (e.g., a function of the measured energy consumption). It then chooses an action from the set of available actions (e.g., a power configuration in this work) and uses the action to configure the environment (e.g., the hardware). The environment will then move to a new state, and the reward associated with the transition is determined. The goal of the RL agent is to learn a policy that maximizes the expected cumulative reward, i.e., the overall energy saving in this work.

In this work, we choose to use a policy network to configure the hardware because a policy network can directly choose what action to take (i.e., what hardware configuration to apply in this work) given the current system state. This formulation naturally fits our problem settings for predicting a single hardware configuration. We learn and update the network using the recently proposed double Q-learning [31], [49], a value-based RL algorithm to explore the discrete optimization space of our problem. Double Q-learning is shown to be more effective than the traditional Q-learning algorithm [50] for RL training, because it can avoid the local optimal due to the overestimation of the reward.

2.3 Motivation Examples

To demonstrate the importance of uncore frequency scaling for power management, we run AMG and miniQMC from the ECP proxy applications suite [51] on a server with two Intel Xeon Gold 6230 processors (2×20 cores, 2×40 threads).

Figure 2 shows the power consumption given by a `baseline` for setting the power cap to the chip's TDP, using the Linux `ondemand` CPU frequency governor for core frequency setting and CPU firmware for uncore frequency scaling. Here, we obtain the best-static configuration (denoted as `static-best`) by profiling all static combinations of the chip's power cap options and uncore frequency settings. The x-axis of the diagram represents the timeline of program execution, and the y-axis represents the average power consumption of each processor. In this experiment, we use the same power configuration throughout the entire program execution, but later we will describe how our approach can dynamically adjust the power configuration.

As can be seen from the diagram, the baseline power management scheme leaves significant room for improvement. For the memory-intensive AMG, the static-best con-

figuration is to cap the processor power at 85 W (40 W below the TDP of 125 W) and set the uncore frequency to 2.2 GHz (0.2 GHz below the default uncore frequency of 2.4 GHz). Such a strategy can save power consumption by 28% at the cost of a minor program slowdown of less than 2%. If we only apply the power cap (`powercap-only`) but let the CPU control the uncore frequency, we can achieve a similar execution overhead as `static-best`, but the CPU will consume 5 W more power. This suggests that it is important to adapt the uncore frequency to maximize energy saving. If we only statically adjust the uncore frequency (`uncore-frequency-only`), we can only achieve a modest energy saving compared to the baseline. The results demonstrate the importance of combining power cap and uncore frequency together for power optimization.

If we now consider the compute-intensive miniQMC application, we see that the `powercap-only` strategy is inadequate for achieving energy saving without significantly compromising the performance. This is not surprising as miniQMC is a CPU-bound application that will suffer from lower CPU core clock frequency imposed by a low power budget. However, because this application's execution time is not dominated by data communications, we can lower the uncore frequency (`uncore-frequency-only`) to reduce the energy consumption by 13%, without performance slowdown. Interestingly, by lowering the uncore frequency, we allow the processor core to occasionally run at a higher frequency while still staying within the chip's power budget, leading to a slightly faster program running time.

This experiment highlights the benefits of combining power capping and uncore frequency scaling for power management. It also shows that the default power management scheme implemented by the operating system and the CPU firmware is ineffective in power optimization. Prior methods based on analytical models are also undesired as they rely on assumptions of the system behavior, which can be too simple for real-world applications [52]. Supervised learning methods avoid the drawbacks of analytical models by learning from empirical observations, but prior work in this area uses a single power configuration throughout the application execution, which cannot adapt to the program phase change. Later in Section 5.2, we show that the state-of-the-art online power management scheme [10] based on machine-learning-based DVFS can give poor performance as it ignores the impact of uncore frequency scaling. Our work is designed to overcome these weaknesses. As we will show later, by dynamically adjusting the hardware configuration throughout the program execution, our approach can achieve better performance than the default configuration.

3 Our Approach

3.1 Overview

Our work optimizes the power consumption of multi-cores by dynamically adjusting the maximum power limit to be used by the chip and configuring the frequency of the uncore. Our goal is to reduce the CPU chip's energy consumption without significantly compromising the application response time. At the core of our approach is an RL-based online power management system that monitors

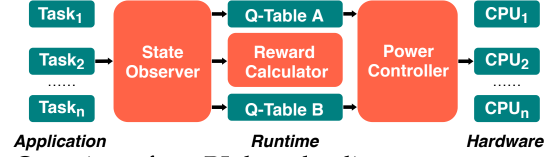


Fig. 3: Overview of our RL-based online power management system framework.

the state of the application and the system to configure the hardware on a per program per CPU basis.

As depicted in Figure 3, our RL framework consists of three components. The system state observer monitors the program behavior through lightweight hardware performance counter measurements. It uses the hardware performance counter information, combining with the historical information and statistical analysis, to detect the phase change of alls programs running on a CPU. The observer also measures the energy consumption of the chip using hardware energy accounting mechanisms (Section 2.1). Based on the energy measurement, the reward calculator computes the rewards of the recently applied power configuration. The measurement is used to update the Q tables for the given power configuration. The Q tables (Section 3.2) are then used to choose the power configuration to be passed to the power controller to configure the power cap and the uncore frequency. Note that we take a decentralized approach by running a system state observer on each of the computing nodes in a distributed environment. The RL system reconfigures the hardware if a program phase change is detected on a local computing node.

3.2 Problem Formulation

Our power management scheme determines the hardware configuration to use based on the current system state. We use the hardware performance counter readings to define the system state. Specifically, in this work, we consider two hardware performance events, the instruction per cycle (IPC) and the misses per operation (MPO), which are commonly available on modern HPC multi-core design (see Section 3.4). In this work, we use the Linux `perf` [53] profiler to read the IPC- and MPO-related events.

For a sequence of system states, $S = \{s_1, s_2, ..., s_n\}$, the RL system needs to take a sequence of actions, $A = \{a_1, a_2, ..., a_m\}$ to achieve the optimization goal. For example, one can divide the IPC values (between 0 and 1.0) into ten regions where each region corresponds to a level of instruction parallelism that corresponds to a system state; for each IPC state, we then record the reward (a transformation of the measured energy consumption) when using an uncore frequency. We use the measured short-term reward to update the Q tables - a data structure used to calculate the maximum expected future rewards for action at each state. Once the Q tables are populated, we can then choose a frequency that gives the maximum total reward for a given IPC state.

3.3 Updating Q Tables

Algorithm 1 outlines our RL-based power management scheme. To begin with, we initialize the Q tables with zero

Algorithm 1: RL-based power management

Input: Test set T
Output: Action a
1 Initialize state s, threshold ϵ_1, threshold ϵ_2, learning rate α, discount factor γ;
2 **for** $i = 1, 2, \ldots, n$ **do**
3   **for** $j = 1, 2, \ldots, m$ **do**
4     $Q^A[s_i, a_j] \leftarrow 0$;
5     $Q^B[s_i, a_j] \leftarrow 0$;
6   **end**
7 **end**
8 Execute test set T;
9 **while** *s is not terminal* **do**
10   Observe system workload w, Estimate the energy consumption (last period E;
11   Calculate state s', based on w;
12   Generate random number Θ_1 ;
13   **if** $\Theta_1 > \epsilon_1$ **then**
14     $a \leftarrow max(Q^A(s,a), Q^B(s,a))$;
15   **else**
16     $a \leftarrow random$;
17   **end**
18   Take action a;
19   Calculate reward r, based on E;
20   Generate random number Θ_2 ;
21   **if** $\Theta_2 > \epsilon_2$ **then**
22     $a^* = argmax_a Q^A(s', a)$;
23     $Q^A(s,a) \leftarrow Q^A(s,a) + \alpha(s,a)[r + \gamma Q^B(s', a^*) - Q^A(s,a)]$;
24   **else**
25     $a^* = argmax_a Q^B(s', a)$;
26     $Q^B(s,a) \leftarrow Q^B(s,a) + \alpha(s,a)[r + \gamma Q^A(s', a^*) - Q^B(s,a)]$;
27   **end**
28   $s \leftarrow s'$;
29 **end**

rewards. We then update the Q tables according to the steps described as follows.

Our state observer periodically takes performance counter readings and uses the measurement to detect program phase changes. If a new program phase is detected, the state observer will record the energy consumption of the system given by the Linux interface (Section 2.1). The reward calculator will then compute the corresponding reward for the currently used power configuration. This is described in more detail in Section 3.4.

Based on the system state representation, the power controller chooses the CPU power budget and the uncore frequency that gives the biggest Q value (i.e., the estimated cumulative reward) according to the Q tables. From time to time, the power configuration will be chosen at random to avoid the system to be trapped in a local optimal. This is done through a ϵ-greedy mechanism that quickly explores the state space. We note that the parameter ϵ is configurable, where a higher value means the greedy algorithm is triggered more often.

Finally, the two tables are updated with different sets of experience samples. Therefore, in each iteration, only one Q table is randomly updated. Then, as shown in Figure 4, each Q table is updated with a value from the other Q table (line 22 in Algorithm 1). This is an unbiased estimate for the value of this action. We use the commonly used Bellman equation [50] to update the Q tables (line 23 in Algorithm 1). For example, in Figure 4, when the measured IPC falls into the "0.1" region, table B is updated with the corresponding values from table A to avoid overestimation caused by Q-learning strategy.

IPC \ Action	1.2GHz	**1.8GHz**	2.4GHz
0.1	0.45	**0.66**	0.82
0.2	0.36	0.54	0.56
……			
1.0	0.22	0.66	0.32

Q-Table A

IPC \ Action	1.2GHz	**1.8GHz**	2.4GHz
0.1	0.55	~~0.88~~ → 0.7	0.76
0.2	0.34	0.52	0.43
……			
1.0	0.21	0.64	0.35

Q-Table B

Fig. 4: Example of how the Q table is updated using feedback from the other table.

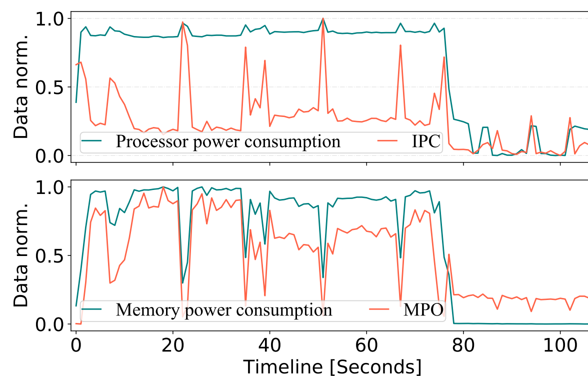


Fig. 5: Phase changes in miniAMR.

3.4 Detect Phase Changes

We represent the system state in a vector of the two hardware performance counter values. Our rationale for choosing these two metrics are justified as follows. Many programs phases belong to one of the two categories according to their computation characteristics: compute-bound and memory-bound. For compute-bound workloads, the commonly used performance indicator is the IPC [54]. A high IPC indicates the program spends most of the time on CPU processing. For memory-bound workloads, we use the MPO [54], [55] to measure how frequently the program access the memory. This metric is derived by normalizing the last level cache misses to the number of instructions measured. As the MPO value increases, the workload becomes more memory-bound because the number of memory accesses per operation grows. In addition, we discretize these performance indicators to obtain a finite state space. Therefore, a drastic change (encoded in the Q tables like Figure 4) in the $< IPC, MPO >$ state vector indicates a phase change.

Running example. Figure 5 shows the energy profile and IPC and MPO readings of miniAMR [51]. To aid clarify, we normalize the measurements by scaling the value to the $[0, 1]$ range using the minimum and maximum values seen during profiling. We can see ten distinct phases where performance counter values change drastically. These phases include short compute-bound phases characterized by high CPU energy consumption, high IPC, low DRAM energy, and low MPO, which occur between relatively longer memory-bound phases characterized by low CPU energy, low IPC, high DRAM energy and high MPO. As a result, the transition from a memory-bound phase to a compute-bound phase can be identified by a rise in IPC or a decline in MPO.

TABLE 1: Hardware platforms and RL settings

Configuration	Platform	Min	Max	RL Step
Power capping	CascadeLake	95 W	125 W	5 W
	Haswell	90 W	120 W	5 W
Uncore frequency	CascadeLake	1.8 GHz	2.4 GHz	0.1 GHz
	Haswell	1.8 GHz	3.0 GHz	0.2 GHz

3.5 Reward Functions

Like all RL systems, we need to have a reward function to compute the instantaneous reward after applying a hardware configuration. Initially, we consider a simple reward function by negating the power consumption measured in each scheduling window - $R_1(s,a) = -power$, where s and a are the state and the corresponding action respectively. This reward function allows us to penalize a configuration that leads to high power consumption, but it ignores the impact on the performance.

To better control the trade-off between energy and performance, we then turn to consider $(IPC)^2/W$, the square of instructions per cycle divides power. In essence, this is an inverse EDP formulation (i.e., $Energy \times Delay$), where we compute the delay as average execution time per instruction. As we are targeting HPC workloads, we would like to give a higher penalty for a longer delay. This results in the third candidate reward function $(IPC)^3/W$, which is essentially an inverse of ED^2P, computed on a per-instruction basis. To make sure the rewards are monotonic increasing, we multiply $(IPC)^2/W$ and $(IPC)^3/W$ by a constant parameter, c (where $c >= 10$). This modification gives us two additional candidate reward functions, $R_2(s,a) = (c \times IPC)^2/W$ and $R_3(s,a) = (c \times IPC)^3/W$.

In Section 5.6, we empirically show that the third reward function, $R_3(s,a) = (c \times IPC)^3/W$, gives the best overall performance and hence is our chosen reward function.

3.6 Complexity Analysis

Our Q tables map a given system state to a power configuration. Identifying a power cap and uncore frequency for a given state from a Q table requires comparing M candidate actions (that are associated with the state) to find the configuration that gives the maximum Q value. The number of candidate actions, M, for a state is constant and depends on the configuration knobs (e.g., the range of power limits and the interval between two settings) provided by the underlying hardware. Hence, the time complexity of our approach is constant, $O(1)$. The space complexity of our scheme is a function of the Q table sizes and the number of computing nodes (as we deploy a copy of the Q tables in each computing node - see Section 3.1). This comes to $O(N * M * |node|)$ for the space complexity, where N and M are the numbers of states and actions, respectively.

4 Experimental Setup

4.1 Platforms and RL Configurations

We evaluate our approach by applying it to two HPC clusters with different Intel CPU architectures, listed in Table 1. Our main evaluation platform consists of four CascadeLake computing nodes. Each node has two 20-core Intel Xeon Gold 6230 processors (160 cores for 4 nodes) and 256 GB memory. The cluster supports 13 uncore frequency levels, from 1.2 GHz to 2.4 GHz, with a step of 0.1 GHz. The power limits of the entire chip can be configured from 65 W to 125 W (TDP). We remark that most of the experiments were conducted on a CascadeLake node except for the scalability experiment presented in Section 5.5. To evaluate the scalability of our approach, we also apply our approach to a second, 16-node computing cluster with Intel Haswell CPUs. Each node of this cluster has one 12-core Intel Xeon E5-2678 v3 processor and 48 GB of RAM. This cluster supports 19 uncore frequency levels, from 1.2 GHz to 3.0 GHz, with a step of 0.1 GHz. The power limits of the entire chip can be configured from 61 W to 120 W (TDP). Note that we set the minimum value of the hardware configurations to the median value provided by the hardware knobs, as using a lower setting has a severely negative impact on the application performance.

TABLE 2: Benchmarks used in our evaluation

Benchmark	Suite	Version	Use	Problem Size
CG	NPB	MPI	Training	A/B/C/D
EP	NPB	MPI	Training	A/B/C/D
FT	NPB	MPI	Training	A/B/C/D
IS	NPB	MPI	Training	A/B/C/D
LU	NPB	MPI	Training	A/B/C/D
MG	NPB	MPI	Training	A/B/C/D
BT	NPB	MPI	Training	A/B/C/D
SP	NPB	MPI	Training	A/B/C/D
LULESH	LLNL	MPI	Testing	30*30*30
miniAMR	ECP	MPI	Testing	6*6*6
HPCCG	ECP	MPI	Testing	200*300*100
miniQMC	ECP	OpenMP	Testing	2*2*2
NekBone	ECP	MPI	Testing	25*2*1
AMG	ECP	MPI	Testing	180*180*180
CoMD	ECP	MPI	Testing	100*100*200
miniFE	ECP	MPI	Testing	500*500*500
Sweep3D	DOE	MPI	Testing	720*360*360
Keras-CNN	BigDL	Spark	Testing	2*5*32*32
LeNet-5	BigDL	Spark	Testing	2*5*32*32

4.2 Systems Software and Sampling

All our evaluation system runs Ubuntu version 16.04 with Linux Kernel 4.15.0. As stated in Section 2.1, we use the `perf` profiler to sample the hardware performance events and the `msr-safe` module to take a reading of the power consumption (given by the RAPL-related MSRs) every 3 seconds. We find that this sampling interval can give accurate energy consumption measurement and allow our RL system to quickly react to the change of program behavior. However, the sampling window can be changed by the user without affecting the working mechanism of our approach in the see also Section 5.7. All the code for monitoring and controlling power is written in C, and is deployed to each node independently. So each node's phase change detection and power configuration are carried out independently.

4.3 Benchmarks

We use 19 parallel benchmarks in our evaluation. These benchmarks represent a wide range of application domains, as listed in Table 2.

Training. Our RL system is trained on the NAS parallel benchmark suite using input classes A, B, C, and D. According to the study in [27], EP is a compute-intensive benchmark, BT, SP, and LU are last level cache-bound

benchmarks, and MG, FT, IS, and CG are memory-bound benchmarks. During training, the RL algorithm learns and updates the Q tables using the measured reward for a power setting under an observed system state (Section 3.4).

Testing. To evaluate the generalization ability of our RL system, we also apply the trained system to 11 benchmarks that are not seen at the training stage. These testing benchmarks include AMG, CoMD, miniFE, miniQMC, HPCCG, NEKbone, and miniAMR from the ECP proxy applications suite [51], LULESH from the LLNL Proxies [56], Sweep3D [57], LeNet-5 [58] and Keras-CNN [59]. Specifically, Sweep3D is a core algorithm used by the DOE's accelerated strategic computing initiative application, and LeNet-5 and Keras-CNN contain the key algorithms used in many deep learning workloads. We use the MPI version of all benchmarks except for miniQMC where use the OpenMP version for single node evaluation and the MPI version running across distributed computing nodes for scalability evaluation (Section 5.5).

4.4 Performance Report

To measure execution time and energy consumption, we run each test case repeatedly until the 95% confidence bound per application per input is smaller than 5%. We then report the average performance across test runs.

4.5 Baseline Scheme

We report energy saving and performance degradation by comparing to a `baseline` scheme, for which we set the power cap to the thermal design power (TDP) of the hardware and let the Linux `ondemand` CPU frequency governor to adjust the CPU core frequency and the CPU firmware modulates the uncore frequency.

4.6 Competitive Schemes

We compare our full implementation against three prior approaches and two implementation variants.

Prior works: We compare our approach against the following three online power management systems [10], [43], [44]:

CoPPer. This implements a feedback controller for power optimization [43]. It uses hardware power capping to meet application performance requirements while achieving energy efficiency. To detect application phase changes, CoPPer requires the user to supply the job latency target, and it also requires the application to measure its own performance progress. By contrast, our approach does not require user involvement and is transparent to the running application.

GEOPM. The Global Extensible Open Power Manager (GEOPM) [44] uses a tree-hierarchial strategy to perform runtime power optimization across distributed computing nodes. GEOPM provides plugins to monitor the progress of current tasks to identify execution bottleneck and adjusts the CPU frequency to achieve load balance among tasks. Its current implementation does not provide an RL-based power management strategy like ours.

RL-based scheme. The closely related work presented in [10] uses double Q-learning to dynamically adjust the CPU frequency for power optimization. It detects the current system phase through the time slack between the CPU idle time and the wall time of program execution. Unlike our approach, this scheme does not explicitly consider the uncore activities.

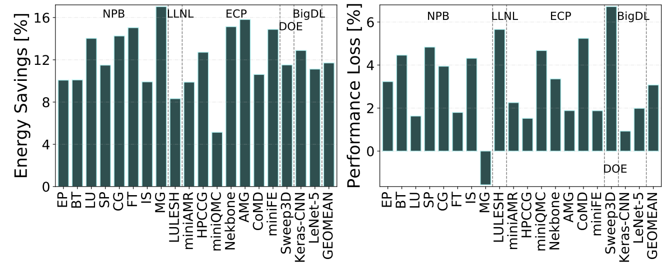


Fig. 6: Energy savings and the resulting slowdowns on a single CascadeLake node with respect to the baseline power management scheme (Section 4.5).

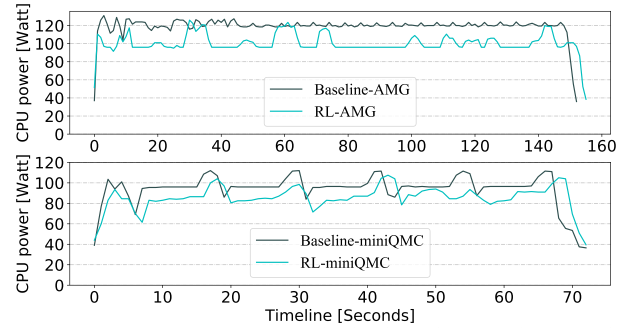


Fig. 7: Power consumption with the baseline and our online power management system.

Implementation variants: We also compare our full implementation to two variant implementations of our scheme:

Powercap. For this method, we apply our RL system to dynamically adjust the CPU power budget based on the program phase change but let the CPU firmware determine the uncore frequency.

UFS. For this implementation, we apply our RL system to only adjust the uncore frequency at each detected program phase change. We use the Linux `ondemand` governor to determine the CPU frequency and set the power cap to TDP.

5 Experimental Results

In this section, we first present the overall results of our approach using all benchmarks (Section 5.1), showing that our approach achieves consistently good performance across evaluated benchmarks. Then, in Section 5.2, we compare our approach against the five alternative schemes described in Section 4.6. Next, we evaluate our approach in scenarios where we run multiple MPI processes of the sample program on a single node (Section 5.3), and where we mix multiple programs on a single node (Section 5.4). We then extend our evaluation to the second computing cluster to evaluate the scalability (Section 5.5) before providing analysis on our design choices in Sections 5.6 and 5.7.

5.1 Overall Results

As can be seen from Figure 6, our training strategy is effective across benchmarks. When the NAS training bench-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3092270, IEEE Transactions on Parallel and Distributed Systems

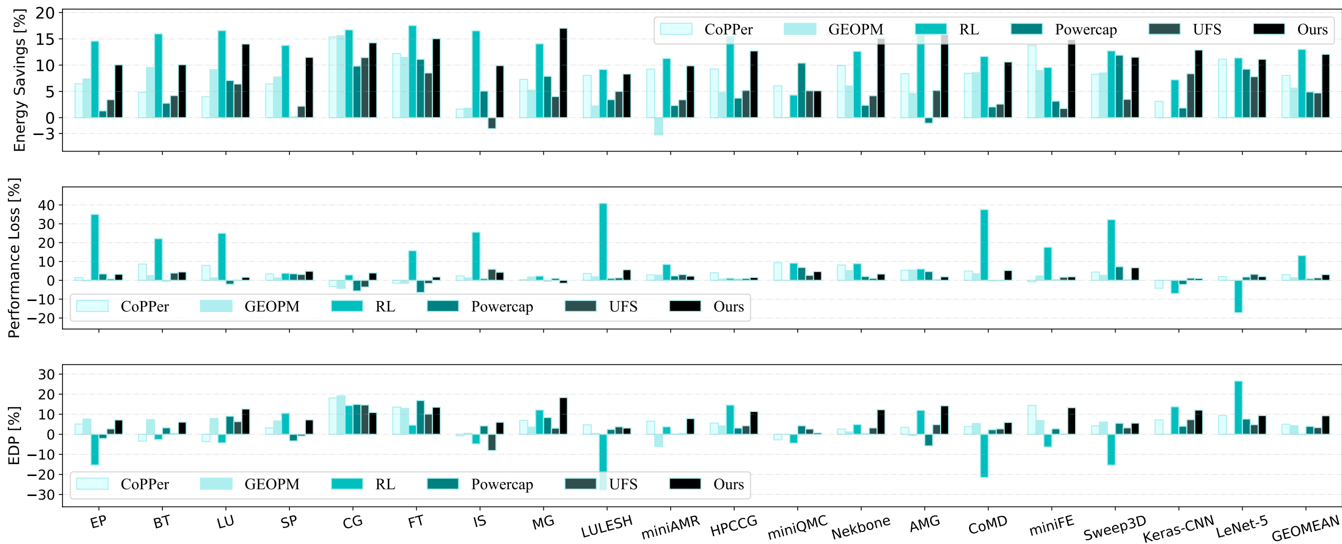


Fig. 8: Energy savings, the resulting slowdowns and EDP of different power management approaches with respect to the baseline power management scheme (Section 4.5).

marks are used in testing, we achieved an average energy saving of 12.5%, with an average performance slowdown of less than 3%. When tested on unseen benchmarks, our approach gives an average energy saving of 12% with less than a 3% slowdown in application performance. The stable performance is because the Q values learning from the training are generally portable across benchmarks, and our RL system can update the Q values using runtime observations. This shows the good generalization ability of our scheme. For most test cases, our approach achieves energy savings with marginal performance degradation. Notably, our approach delivers energy savings for both compute-bound and memory-bound applications. For instance, applications like EP, BT, SP, MiniAMR, miniQMC and Nekbone have longer CPU-bound phases than others. Our approach achieves energy reduction by running the application at a higher power cap with a lower uncore frequency for these applications. By contrast, our approach chooses a lower power cap for memory-bound applications to reduce the core frequency but increase the uncore frequency as the memory access is the bottleneck. Such an adaptive scheme allows our approach to reduce energy consumption for most test cases at the cost of marginal performance degradation.

Working examples. Figure 7 takes a close look at AMG and miniQMC seen earlier in Section 2.3. It shows the average CPU power consumption resulting from our approach against the baseline scheme described in Section 4.5. Two observations can be derived from these experimental results. Firstly, with our approach, the CPU spends more time at a lower power state than the baseline. This is observed for both the memory-intensive AMG benchmark and the compute-bound miniQMC benchmark. Secondly, our approach can adapt to program phase changes by dynamically adjusting the hardware configuration. Specifically, for miniQMC that incur frequent phase changes, our approach does not always stay at the lower power cap stage. Instead, it detects the phase change and adjusts the power cap and uncore frequency accordingly, demonstrating the adaptiveness of our scheme.

5.2 Compare to Alternative Schemes

As can be seen from Figure 8, our approach gives the best trade-off between energy saving and performance loss when comparing to the five alternative power management methods described in Section 4.6. For example, our scheme gives a similar, modest performance loss of less than 3% on average when compared to CoPPer, but improves the energy saving by 4% over CoPPer. GEOPM is conservative in trading performance for energy reduction compared to other approaches. As a result, it gives less slowdown, less than 2% on average. However, GEOPM is over-conservative, which only delivers less than half of the energy reduction give by our approach. Finally, although the RL-based alternative scheme [10] gives slightly better energy reduction than our scheme, this often comes at a significant performance penalty – up to 40% for some benchmarks. If we now consider the improvement on the EDP, a higher-is-better metric for quantifying the trade-off between performance loss and energy saving. Our approach achieves a better trade-off between performance and energy-saving on most of the benchmarks compared to alternative schemes. While the state-of-the-art RL-based energy optimization scheme (i.e., RL on the diagram) can achieve a higher EDP over our approach on four benchmarks, it can give poor EDP on compute-bound benchmarks, leading to an overall degradation on the EDP.

Our approach for combining power capping with uncore frequency scaling gives more benefit than that of a single operation. For instance, the prior RL-based method [10] and our powercap-only (`powercap`) implementation variant only enforces power cap but ignore the uncore frequency. Both strategies give an average energy saving of 4%, with the average performance loss is 0.8% and 1.3%, respectively. Compared to these schemes, our approach give 3x more energy saving but with a similar performance slowdown. By dynamically scaling down the uncore frequency, we open

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3092270, IEEE Transactions on Parallel and Distributed Systems

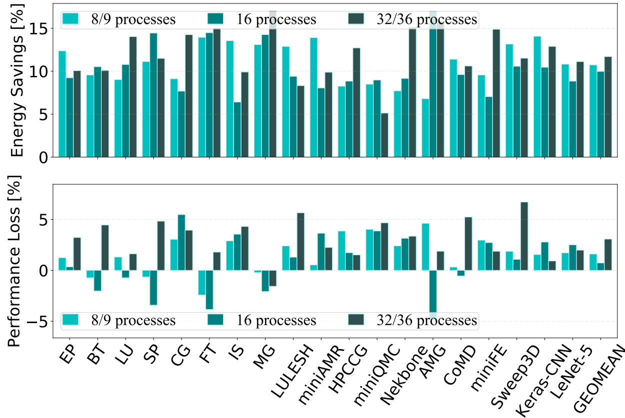


Fig. 9: Energy savings and the resulting slowdowns when using different numbers of MPI processes on a single node, with respect to the baseline (Section 4.5).

TABLE 3: Multi-application Workloads

Name	Applications	Name	Applications
$mix1$	AMG, IS, MG	$mix2$	AMG, MG, miniFE
$mix3$	EP, miniQMC, Nekbone	$mix4$	IS, miniFE, miniQCM

the availability to run the core to higher frequencies because a low uncore frequency enables the saved uncore power budget to be used to boost the frequency of the processor cores. The increased core frequency, in turn, allows us to deliver similar performance for compute-bound applications while lowering the overall energy consumption. Overall, our approach can better trade application performance for energy reduction. If we now consider the EDP and ED^2P metrics, our approach improves EDP and ED^2P by 4% and 10% on average when compared to the best-performing alternative method.

5.3 Impact of Concurrent Processes on a Single Node

One way of improving the system utilization is to run multiple programs or processes on a single computing node. In this experiment, we study the impact of running concurrent MPI processes on a single node by using a single RL system to perform power optimization across multiple processes. Intuitively, having more concurrent processes increases the complexity of the optimization space.

Figure 9 shows how the number of concurrent processes affects the performance of our approach. In this experiment, we vary the MPI processes used for each application by running different MPI processes as given in the legend. The number of MPI processes has little impact on the power management system. The power management system saves energy by 11%, 10%, and 12%, and reduces the completion time by 1.6%, 0.7%, and 3% for the three process numbers, respectively. The observation indicates adaptability of the power manager to the application parameters changes, thus leading to energy-efficient power management.

Our approach not only saves energy but also improves the application performance in some cases. For instance, when SP is executed, the system reduces the energy consumption by 14% with a performance improvement of 3% over the baseline. Similarly, when AMG is executed, it achieves an energy saving up to 17% with a performance improvement of 5%. The improved performance is because when our approach lowers the uncore frequency under a power cap, the saved energy budget can be used to boost the processor core clock frequency for fast computation.

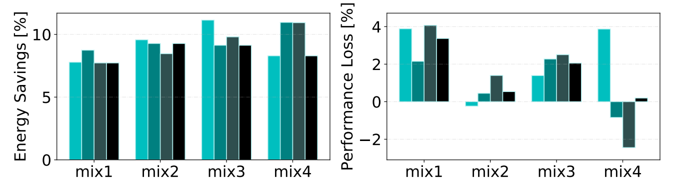


Fig. 10: Energy savings and the resulting slowdowns on multi-application workloads with respect to the baseline (Section 4.5).

5.4 Impact of Multi-Application Workloads

In this experiment, we evaluate our approach using multi-application workloads. The benchmarks used include both computation-bound and memory-bound workloads. We create the workload mix by selecting applications from these two classes, as listed in Table 3. Specifically, we create four separate mixes, each consisting of three applications. Each of the mixtures is given the name $mixN$. For the first two mixes, the applications are drawn from memory-bound workloads. The mix, mix3, is when all applications are compute-bound. Finally, the applications in mix4 include three applications from two categories. During experiments, we launch all benchmarks at the same time. We assume that the application runtime knows that they are running with other applications. Hence, each runs with only 8 processes, so that the total number of active processes is no more than the number of actual cores. We pin the MPI processes to CPU cores by using the "–cpu-set" and "–bind-to core" runtime options when launching an MPI program. The former option tells which set of CPU cores the processes of an MPI program can run on per node, and the latter binds individual processes to cores within the given CPU set.

As shown in Figure 10, our approach still gives considerable energy saving in a multi-program setup, although the saving is relatively smaller than in the single-application scenario. This is due to the complex interactions among resource competition and synchronization, which create a larger and more complex optimization space for the RL system to explore. Nonetheless, our automatic scheme still achieves over 8% energy reduction without user involvement. For some application mixes, it gives speedups (i.e., a negative performance loss value in Figure 10) by allowing the cores to run on a higher frequency using a lower uncore frequency.

5.5 Scalability

So far, all the evaluations were performed on a single computing node from the CascadeLake cluster. In this experiment, we evaluate our approach in a distributed computing environment by applying our approach to the 4-node and 16-node clusters described in Section 4.1. On each computing node, we run a decentralized RL system to monitor and perform power optimization at the node

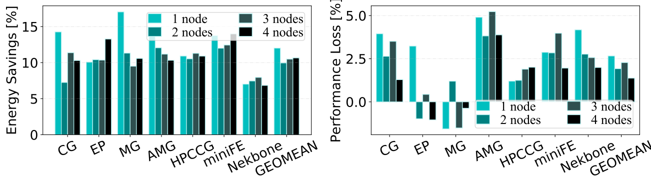


Fig. 11: Energy savings and the resulting slowdowns on Cascadelake cluster with respect to the baseline (Section 4.5).

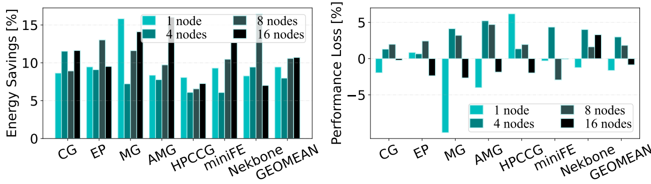


Fig. 12: Energy savings and the resulting slowdowns on Haswell cluster with respect to the baseline (Section 4.5).

level. In each computing cluster, we vary the number of computing nodes used in our evaluation. Figures 11 and 12 show the result on the 4-node CascadeLake and the 16-node Haswell respectively. Like our previous evaluations, we normalize the results to the baseline power management scheme described in Section 4.5.

Our approach exhibits good scalability and portable performance across the two computing clusters. It delivers similar energy reduction with modest performance penalty as the number of computing nodes increased. For the CascadeLake cluster, when using a single computing node, the average energy reduction and performance slowdown is 12% and 2.5%, respectively. In comparison, when using 2, 3 and 4 nodes, the energy reduction is 10%, 10.5%, and 11%, respectively, and the performance loss is 2%, 2.5%, and 1.5%, respectively. We also observe a similar trend on the 16-node Haswell cluster. On a single node, the average energy reduction is 10% on the Haswell cluster. In comparison, the energy reduction is between 8% and 11% when using more than one computing node. Furthermore, the performance impact is more or less the same when using a different number of computing nodes on this cluster.

5.6 Impact of Reward Functions

We now compare the three RL reward functions described in Section 3.5. The results in Figure 13 suggest that our chosen reward function $(c \times IPC)^3/W$ gives the best overall trade-off between energy reduction and performance.

Firstly, using the negative number of power consumption (i.e., `-power`) as the reward function leads to the least energy saving. This result may seem to be counterintuitive as this scheme aims to lower energy consumption regardless of the program slowdown. However, the relatively low energy reduction and performance slowdown are because, in our evaluation, we set the minimum power cap and uncore frequency to the medium values supported by the hardware (Section 4.1). This setup limits the ability of this reward function to further lower the frequency to achieve better energy saving (which, in turn, limits the performance slowdown). When lifting this restriction, we found that

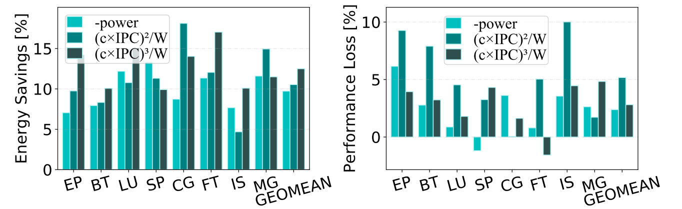


Fig. 13: Energy savings and the resulting slowdowns on different rewards with respect to the baseline.

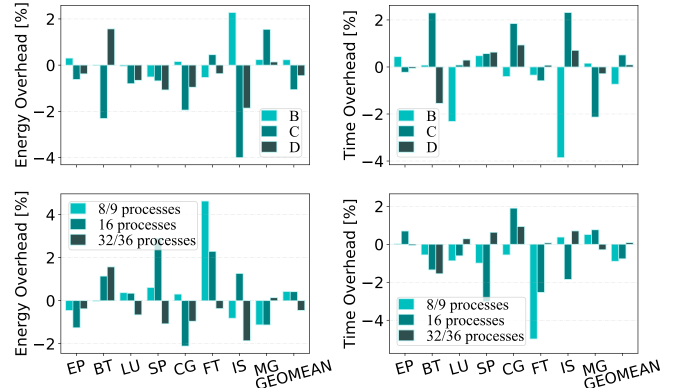


Fig. 14: Energy and time overhead on different problem sizes and processes w.r.t. the baseline without sampling.

the `-power` reward function can further reduce the energy consumption but at the cost of massive program slowdown.

If we now consider the more balance reward functions $(c \times IPC)^2/W$ and $(c \times IPC)^3/W$ that simultaneously consider energy and performance, we see that $(c \times IPC)^3/W$ finds a better trade-off between energy reduction and performance slowdown. While $(c \times IPC)^2/W$ can sometimes give higher energy saving over $(c \times IPC)^3/W$, it can lead to significantly higher performance loss. For this reason, we choose $(c \times IPC)^3/W$ as our reward function.

5.7 Performance Counter Sampling Overhead

We now evaluate the impact of our sampling window for detecting phase changes. Figure 14 shows the overhead to the energy consumption and execution time by normalizing the resulting performance given by our sampling window to a baseline (Section 4.5) that does not incur sampling overhead. Here, a negative value means our approach actually reduces the energy consumption or improves performance (by making the program runs faster). As can be seen from the diagram, our choice of sampling window has a negligible negative impact on energy consumption and performance. Figure 15 shows the average sampling overhead for energy and performance averaged across our benchmark settings when we vary the sampling window size. We report the data by averaging the overhead across all our test benchmarks and datasets. Note that the minimum sampling window provided by *perf* is 3 ms. As can be seen from the diagram, performance counter sampling has little impact on the application performance and energy overhead. We choose a sampling window of 3 seconds in this work as we found it to be sufficient in detecting

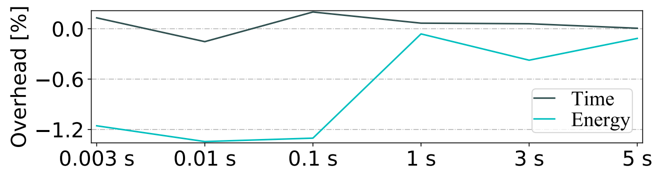


Fig. 15: Overhead of different sampling window sizes with respect to the baseline without sampling.

program phase changes. However, this parameter can be easily reconfigured by the user.

6 Discussion and Future Work

Our approach is among the first attempts in applying RL to optimize CPU energy consumption by simultaneously considering the power cap and uncore frequency during dynamic runtime. Naturally, there is room for improvement and further work.

Model interpretability. Machine learning techniques, in general, have the problem of relying on black boxes. This is just as true for our RL-based method. One way to gain insight into why the model makes a decision is to train an interpretable model (or the so-called surrogate models) like linear regressor [60] or a Markov Decision Process (MDP) with a value function [50] to approximate the predictions of the underlying black-box model. We view it as an exciting future challenge to find ways to better interpret the working mechanism of an RL-based power management scheme. Can we correlate the decision to the high-level system status and program behavior? Can we quantify why a local optimal may not lead to a better long-term reward for a long-running program?

Multi-tasking environment. Programs often do not run in isolation and have to compete for the shared computing resources with other concurrently running programs [61], [62], [63]. Our current implementation performs power optimization on the system level across multiple running programs. It would be interesting to extend our approach to apply individual program phase changes to derive hardware configurations on a per-program, per-core basis. For example, a memory-bound task can run on a CPU core with a low frequency, and the saved power budget can be used to increase the core frequency where a computation-intensive application runs on. Such capability may require the hardware to support individual power domains at the processor core level. We leave this as our future work.

Global optimization. Our current implementation deploys an RL system to each computing node, and each deployed RL system makes decisions independently on the local node. This decentralized strategy avoids the synchronization overhead when processes running on different nodes do not perfectly align in steps. It also allows our approach to scalable to a large, distributed environment. It would be interesting to introduce some lightweight schemes to coordinate the executions and optimizations across distributed computing nodes to improve the overall systems throughput and energy efficiency. For example, our approach can benefit from the hierarchical optimization framework of GEOPM [44], by using it to propagate the information in a tree-like computing node structure and use the feedback to coordinate the optimization across computing nodes.

7 Related Work

Our work builds upon the following past foundations but is quality different from each.

Online power management. Numerous online power management approaches have been proposed [7], [27], [28], [29], [44], [64]. Conductor accelerates the application's critical path to reduce the waiting time and energy consumption of non-critical execution paths (or threads) [64]. GEOPM is an open-source power optimization framework [44]. It organizes the distributed computing nodes in a tree-like hierarchy to coordinate the power optimization decisions across computing nodes. GEOPM allows a new energy management strategy to be implemented as a plugin. All these methods use expert-crafted heuristics, which are expensive to build as they require expert insights into the workloads and the computing system. Our approach reduces human involvement by directly learning how to perform energy optimization through empirical observations and environment interactions. Given the diverse set of application workloads and hardware platforms, an automated approach based on empirical observations rather than expert knowledge is more sustainable and scalable.

RL based energy optimization. By using the feedback from the system environment, a machine-learning-based power manager learns to improve its decisions over time [10], [19], [20], [21], [65], [66], [67]. The work presented in [10] is most closely related to our approach, which uses RL to adjust the CPU clock frequency. Unlike our approach, this approach does not model the uncore domain. Moreover, as we have shown in Section 5.2, this approach can also lead to significant violation of performance guarantee. Given a total power budget, PowerCoord employs RL to dynamically adjust the power supply for the CPU and GPUs to maximize the system throughput [19]. Unlike our approach, none of the aforementioned approaches targets uncore frequency optimization. However, techniques like transfer learning [68] and collective learning [21] are orthogonal to our approach.

Dynamic power capping. Most of the existing power optimization methods do not dynamically determine the power budget according to the program behavior [6], [23], [43], [69]. Our previous work [23] uses a machine-learning model to derive a *static* power capping configuration but cannot adapt to the program phase changes. Furthermore, the machine-learned model is frozen after training and hence can give a poor performance for previously unseen workload behavior. Our approach avoids these drawbacks by using RL to continuously update its decisions to adapt to the changes of program workloads and runtime phases. Other works [70], [71], [72], [73] study how different power caps affect the performance of numerical algorithms with different computational intensities, showing the importance of choosing an appropriate power budget at runtime. Our work builds upon these prior studies to propose an automatic approach to perform CPU power optimization by con-

sidering CPU power capping and uncore frequency scaling at the same time.

8 Conclusion

We have presented a reinforcement learning (RL) based approach for online power management, targeting modern high-performance multi-core architectures. Our techniques dynamically modulate the power cap of the multi-core chip and the uncore frequency to match the runtime program behavior. Our work learns how to leverage the hardware power optimization mechanisms from training programs. It then uses the learned knowledge to perform power optimization for new, unseen programs, adapting as needed. Unlike prior machine-learning-based approaches, our approach can adapt to the program phase changes and use runtime feedback to update its decision agent during execution time. We evaluate our approach by applying it to optimize parallel programs running on two distributed HPC clusters. Experimental results show that our approach can reduce the CPU energy consumption by 12% on average, with less than 3% slowdown in the program running times. In certain cases, we can reduce the energy consumption by 17% while accelerating the program execution time by 5%.

Acknowledgment

This work was supported in part by the National Key Research and Development Program of China under Grant agreement 2017YFB0202901, the Key-Area Research and Development Program of Guangdong Province under Grant agreement 2019B010136001, the National Natural Science Foundation of China under Grant agreements 61672186 and 61872294, and the Shenzhen Technology Research and Development Fund under Grant agreement JCYJ20190806143418198. Prof. Zhang is the corresponding author.

References

[1] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina *et al.*, "The opportunities and challenges of exascale computing," *Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee*, pp. 1–77, 2010.

[2] P. Beckman, R. Brightwell, M. Gokhale, B. R. de Supinski, S. Hofmeyr, S. Krishnamoorthy, M. Lang, B. Maccabe, J. Shalf, and M. Snir, "Exascale operating systems and runtime software report," USDOE Office of Science (SC), Washington, DC (United States), Tech. Rep., 2012.

[3] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*. IEEE, 2008, pp. 123–134.

[4] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory power estimation and capping," in *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. IEEE, 2010, pp. 189–194.

[5] A. Devices, "BIOS and kernel developer's guide (BKDG) for AMD family 15h models 00h–0fh processors (2012)."

[6] P. Petoumenos, L. Mukhanov, Z. Wang, H. Leather, and D. S. Nikolopoulos, "Power capping: What works, what does not," in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2015, pp. 525–534.

[7] H. Zhang and H. Hoffmann, "Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques," *The ACM Special Interest Group on Programming Languages (SIGPLAN) Notices*, vol. 51, no. 4, pp. 545–559, 2016.

[8] H. Khaleghzadeh, M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "Bi-objective optimization of data-parallel applications on heterogeneous hpc platforms for performance and energy through workload distribution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 543–560, 2020.

[9] H. Huang, M. Lin, and Q. Zhang, "Double-Q learning-based DVFS for multi-core real-time systems," in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2017, pp. 522–529.

[10] H. Huang, M. Lin, L. T. Yang, and Q. Zhang, "Autonomous power management with Double-Q reinforcement learning method," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1938–1946, 2019.

[11] J. Hofmann, G. Hager, and D. Fey, "On the accuracy and usefulness of analytic energy models for contemporary multicore processors," in *International conference on high performance computing*. Springer, 2018, pp. 22–43.

[12] H.-Y. Cheng, J. Zhan, J. Zhao, Y. Xie, J. Sampson, and M. J. Irwin, "Core vs. uncore: The heart of darkness," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.

[13] E. André, R. Dulong, A. Guermouche, and F. Trahay, "DUF: Dynamic uncore frequency scaling to reduce power consumption," 2020.

[14] D. L. Hill, D. Bachand, S. Bilgin, R. Greiner, P. Hammarlund, T. Huff, S. Kulick, and R. Safranek, "The uncore: A modular approach to feeding the high-performance cores." *Intel Technology Journal*, vol. 14, no. 3, 2010.

[15] V. Gupta, P. Brett, D. Koufaty, D. Reddy, S. Hahn, K. Schwan, and G. Srinivasa, "The forgotten 'uncore': On the energy-efficiency of heterogeneous cores," in *2012 USENIX Annual Technical Conference (ATC)*, 2012, pp. 367–372.

[16] B. Subramaniam and W.-c. Feng, "Towards energy-proportional computing for enterprise-class server workloads," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, 2013, pp. 15–26.

[17] P. Balaprakash, D. Buntinas, A. Chan, A. Guha, R. Gupta, S. H. K. Narayanan, A. A. Chien, P. Hovland, and B. Norris, "Exascale workload characterization and architecture implications," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2013, pp. 120–121.

[18] T. Shi, M. Zhai, Y. Xu, and J. Zhai, "Graphpi: High performance graph pattern matching through effective redundancy elimination," *arXiv preprint arXiv:2009.10955*, 2020.

[19] R. Azimi, C. Jing, and S. Reda, "Powercoord: Power capping coordination for multi-CPU/GPU servers using reinforcement learning," *Sustainable Computing: Informatics and Systems*, vol. 28, p. 100412, 2020.

[20] R. A. Shafik, S. Yang, A. Das, L. A. Maeda-Nunez, G. V. Merrett, and B. M. Al-Hashimi, "Learning transfer-based adaptive energy minimization in embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 6, pp. 877–890, 2015.

[21] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1203–1215, 2018.

[22] L. K. Documentation, "Intel p-state driver," 2016.

[23] 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*, 2020.

[24] H. Jung and M. Pedram, "Supervised learning based power management for multicore processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 9, pp. 1395–1408, 2010.

[25] C. Imes, S. Hofmeyr, and H. Hoffmann, "Energy-efficient application resource scheduling using machine learning classifiers," in *Proceedings of the 47th International Conference on Parallel Processing*, 2018, pp. 1–11.

[26] H. Sayadi, N. Patel, A. Sasan, and H. Homayoun, "Machine learning-based approaches for energy-efficiency prediction and scheduling in composite cores architectures," in *2017 IEEE Inter-*

national Conference on Computer Design (ICCD). IEEE, 2017, pp. 129–136.

[27] N. Gholkar, F. Mueller, and B. Rountree, "Uncore power scavenger: A runtime for uncore power conservation on HPC systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–23.

[28] N. Gholkar, F. Mueller, B. Rountree, and A. Marathe, "Pshifter: Feedback-based dynamic power shifting within hpc jobs for performance," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, 2018, pp. 106–117.

[29] V. Sundriyal, M. Sosonkina, B. Westheimer, and M. S. Gordon, "Maximizing performance under a power constraint on modern multicore systems," *Journal of Computer and Communications*, vol. 7, no. 7, pp. 252–266, 2019.

[30] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal and autonomous control using reinforcement learning: A survey," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2042–2062, 2017.

[31] H. Hasselt, "Double Q-learning," *Advances in neural information processing systems*, vol. 23, pp. 2613–2621, 2010.

[32] P. Zhang, J. Fang, T. Tang, C. Yang, and Z. Wang, "Auto-tuning streamed applications on Intel Xeon Phi," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 515–525.

[33] Z. Wang and M. F. O'Boyle, "Partitioning streaming parallelism for multi-cores: a machine learning based approach," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, 2010, pp. 307–318.

[34] D. Grewe, Z. Wang, and M. F. O'Boyle, "Portable mapping of data parallel programs to opencl for heterogeneous systems," in *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2013, pp. 1–10.

[35] Z. Wang and M. F. O'boyle, "Using machine learning to partition streaming programs," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 10, no. 3, pp. 1–25, 2013.

[36] P. Zhang, J. Fang, T. Tang, C. Yang, and Z. Wang, "Auto-tuning streamed applications on intel xeon phi," in *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21-25, 2018*, 2018, pp. 515–525.

[37] P. Zhang, J. Fang, C. Yang, C. Huang, T. Tang, and Z. Wang, "Optimizing streaming parallelism on heterogeneous many-core architectures," *IEEE Trans. Parallel Distributed Syst.*, vol. 31, no. 8, pp. 1878–1896, 2020.

[38] G. Ye, Z. Tang, H. Wang, D. Fang, J. Fang, S. Huang, and Z. Wang, "Deep program structure modeling through multi-relational graph-based learning," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 2020, pp. 111–123.

[39] W. F. Ogilvie, P. Petoumenos, Z. Wang, and H. Leather, "Minimizing the cost of iterative compilation with active learning," in *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2017, pp. 245–256.

[40] Z. Wang and M. O'Boyle, "Machine learning in compiler optimization," *Proceedings of the IEEE*, vol. 106, no. 11, pp. 1879–1901, 2018.

[41] C. Cummins, P. Petoumenos, Z. Wang, and H. Leather, "Synthesizing benchmarks for predictive modeling," in *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2017, pp. 86–99.

[42] H. Wang, G. Ye, Z. Tang, S. H. Tan, S. Huang, D. Fang, Y. Feng, L. Bian, and Z. Wang, "Combining graph-based learning with automated data collection for code vulnerability detection," *IEEE Transactions on Information Forensics and Security*, 2020.

[43] C. Imes, H. Zhang, K. Zhao, and H. Hoffmann, "Copper: Soft real-time application performance using hardware power capping," in *2019 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 2019, pp. 31–41.

[44] 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 *International Supercomputing Conference*. Springer, 2017, pp. 394–412.

[45] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore, "Measuring energy and power with PAPI," in *2012 41st international conference on parallel processing workshops*. IEEE, 2012, pp. 262–268.

[46] "Linux support for power measurement interfaces." http://web.eece.maine.edu/~vweaver/projects/rapl/rapl_support.html, 2021.

[47] K. Shoga, B. Rountree, M. Schulz, and J. Shafer, "Whitelisting MSRs with msr-safe," in *3rd Workshop on Exascale Systems Programming Tools, in conjunction with SC14*, 2014.

[48] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE circuits and systems magazine*, vol. 9, no. 3, pp. 32–50, 2009.

[49] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[50] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[51] "ECP apps," https://proxyapps.exascaleproject.org/app/, 2020.

[52] K. Fan, B. Cosenza, and B. Juurlink, "Predictable GPUs frequency scaling for energy and performance," in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–10.

[53] A. C. De Melo, "The new linux 'perf' tools," in *Slides from Linux Kongress*, vol. 18, 2010, pp. 1–42.

[54] S. Labasan, "Energy-efficient and power-constrained techniques for exascale computing," *Semanticscholar: Seattle, WA, USA*, 2016.

[55] V. W. Freeh and D. K. Lowenthal, "Using multiple energy gears in MPI programs on a power-scalable cluster," in *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2005, pp. 164–173.

[56] Lawrence Livermore National Laboratory, "LLNL proxy apps suite," https://computing.llnl.gov/projects/co-design/proxy-apps, 2020.

[57] A. Hoisie, O. Lubeck, and H. Wasserman, "Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications," *The International Journal of High Performance Computing Applications*, vol. 14, no. 4, pp. 330–346, 2000.

[58] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[59] N. Ketkar, "Introduction to keras," in *Deep learning with Python*. Springer, 2017, pp. 97–111.

[60] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?" Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[61] M. K. Emani, Z. Wang, and M. F. O'Boyle, "Smart, adaptive mapping of parallelism in the presence of external workload," in *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2013, pp. 1–10.

[62] D. Grewe, Z. Wang, and M. F. O'Boyle, "OpenCL task partitioning in the presence of GPU contention," in *International Workshop on Languages and Compilers for Parallel Computing*. Springer, 2013, pp. 87–101.

[63] Y. Wen, Z. Wang, and M. F. O'boyle, "Smart multi-task scheduling for opencl programs on CPU/GPU heterogeneous platforms," in *2014 21st International conference on high performance computing (HiPC)*. IEEE, 2014, pp. 1–10.

[64] A. Marathe, P. E. Bailey, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski, "A run-time system for power-constrained HPC applications," in *International conference on high performance computing*. Springer, 2015, pp. 394–408.

[65] J. Ren, L. Gao, H. Wang, and Z. Wang, "Optimise web browsing on heterogeneous mobile platforms: a machine learning based approach," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[66] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive deep learning model selection on embedded systems," *ACM SIGPLAN Notices*, vol. 53, no. 6, pp. 31–43, 2018.

[67] J. Ren, X. Wang, J. Fang, Y. Feng, D. Zhu, Z. Luo, J. Zheng, and Z. Wang, "Proteus: Network-aware web browsing on heterogeneous mobile systems," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, 2018, pp. 379–392.

[68] J. Ren, L. Yuan, P. Nurmi, X. Wang, M. Ma, L. Gao, Z. Tang, J. Zheng, and Z. Wang, "Optimise web browsing on heterogeneous mobile platforms: a machine learning based approach," in *IEEE Conference on Computer Communications*. IEEE, 2020.

[69] L. Mukhanov, P. Petoumenos, Z. Wang, N. Parasyris, D. S. Nikolopoulos, B. R. De Supinski, and H. Leather, "Alea: A fine-

grained energy profiling tool," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 1, pp. 1–25, 2017.
[70] A. Haidar, H. Jagode, P. Vaccaro, A. YarKhan, S. Tomov, and J. Dongarra, "Investigating power capping toward energy-efficient scientific applications," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 6, p. e4485, 2019.
[71] A. Haidar, H. Jagode, A. YarKhan, P. Vaccaro, S. Tomov, and J. Dongarra, "Power-aware computing: Measurement, control, and performance analysis for Intel Xeon Phi," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–7.
[72] S. Ramesh, S. Perarnau, S. Bhalachandra, A. D. Malony, and P. Beckman, "Understanding the impact of dynamic power capping on application progress," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 793–804.
[73] L. Riha, O. Vysocky, and A. Bartolini, "Evaluation of DVFS and uncore frequency tuning under power capping on Intel Broadwell architecture." in *2019 International Conference on Parallel Computing (PARCO)*, 2019, pp. 634–643.

Yiming Wang received the MS degree in software engineering from Harbin Institute of Technology, China, in 2019. She is currently working toward the PhD degree in the School of Cyberspace Science, Harbin Institute of Technology. Her research interests include high performance computing, performance optimization, and energy efficiency.

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 in the School of Cyberspace Science at Harbin Institute of Technology, China. His research interests are primarily in parallel computing, distributed computing, cloud and grid computing, and computer network.

Meng Hao received the BS degree in computer science and engineering from Harbin Institute of Technology, China, in 2014. He is currently working toward the PhD degree in the School of Cyberspace Science, Harbin Institute of Technology. His research interests include high performance computing, performance modeling, and parallel optimization.

Zheng Wang is an associate professor with the University of Leeds. His research interests include compilers, programming models, parallel computing, runtime systems, and systems security.