Light Weight RL Based Run Time Power Management Methodology for Edge Devices

Ratnala Vinay[1], Pradip Sasmal[1], Chandrajit Pal[1], Toshihisa Haraki[2], Kazuhiro Tamura[2], Chirag Juyal[2]
Mohamed Amir Gabir Elbakri[2], Sumohana Channappayya[1] and Amit Acharyya[1]
[1]*IIT Hyderabad, Telangana-502285, India*, [2]Suzuki Motor Corporation, Japan
email:{ee21resch11009@iith.ac.in}, {sasmal.pradip, palchandrajit}@gmail.com,
{haraki, k_tamura, juyal_c, mohamed}@hhq.suzuki.co.jp, {sumohana,amit_acharyya}@ee.iith.ac.in

***Abstract*—Run time power management has been a major problem in modern-day edge computing devices. Most of the built-in run-time power managers are not adaptable across a variety of workloads. A similar problem in the domain of High-Performance Computing (HPC) is resolved using Reinforcement learning (RL) approaches like Q-learning which can continuously learn from new workload scenarios. The main bottleneck of implementing Q-learning on edge compute platforms is the large Q-table size and the compute load as the algorithm continuously runs in the background. Compute load by RL continuously running in the background adds a significant amount of overhead on power which needs to be addressed to meet the power constraints for edge devices. As a part of our work, we propose a lightweight Q-learning methodology with intelligent memory management and algorithmic workload management policy. These policies make the algorithm lightweight and fit on edge computing devices. This lightweight Run Time Manager (RTM) eventually helps to bring down the power in run time. Our model has been implemented on the A57 processor of the Jetson Tx2 board which is widely used in a number of edge devices. Our proposed lightweight methodology brought run-time average power saving of 16.63% and the Q-table size decreased by 60% compared to the state-of-the-art edge computation Q-learning approach.**

***Index Terms*—Edge compute devices, Power management, Reinforcement learning, Q-learning, Memory management, Workload management**

I. Introduction

In modern-day compute hardware with the ever-growing complexity of the applications being run on them, methods to reduce the run-time power consumption without compromising the performance have become a challenging task. Most of the systems tend to manage the run-time power management policy by built-in OS methods like performance, power save etc. One common disadvantage of all these OS-based approaches is they are programmed in such a way that they have very less adaptability to unseen loads. This improper optimization for new workload scenarios results in a significant amount of power loss. In the domain of High-Performance Computing (HPC) this problem of adaptability to loads is addressed by implementing Reinforcement learning (RL) based methods specially Q-learning based methods [1]–[7]. These approaches maintain a table called Q-table which stores the rewards corresponding to agent actions. Q-learning-based approaches deploy Application Program Interface (API) based action to compute the reward and train itself. One of the most widely used API-based activation is Dynamic Voltage and Frequency Scaling (DVFS) where dynamically the voltage and frequency are scaled at run time to reach optimal power consumption [8]–[11]. A mix of DVFS and Dynamic Core Switching (DCS) were also proposed [12]–[14].RL based methods have found limited application in the edge compute run time power optimization. The major reasons are the large Q-table sizes and compute load which cannot be handled due to constraints on available hardware resources. State space in Q-learning is large as we are completely blind to the dependency of state variables with reward function. This results in a fairly large amount of memory consumption. Q-learning continuously running in the background to monitor the environment is a major reason for the computational load. This computational load by RL may result in the performance of other active processes getting hampered. Moreover, this load adds an overhead on the power consumed. These problems of limited memory, power and hardware resources are to be addressed to make any RL algorithm fit on an edge device. Few works have tried to address this problem of RL on edge [10], [14] but generic methods have not been proposed to downsize the state space and manage algorithmic workload.

In our proposed methodology we have tried to address the above issues and come up with a lightweight RL methodology in terms of Q-table size and compute load which can fit edge hardware applications. As a part of this work, we introduce memory management and an algorithmic workload management policy.

- Our proposed memory management policy has a pre-processing step which correlates the convergence rate of the reward function with the state variable and selects the most correlated state variable. By this, we replace the multi-variable input state vector with a single variable state vector, resulting in very small state space and reduced Q-table sizes. This approach is independent of the reward function and can help to form one variable state space for any generic reward function.
- The algorithmic workload is mainly contributed by the fact that the RL algorithm is always running to monitor

This work is partially supported by the Ministry of Electronics and Information Technology, Government of India and Suzuki Motor Corporation, Japan

the environment. This issue is addressed by introducing a method called dynamic load-based switching. This method switches off the RL in certain system states and runs a built-in optimization policy. The switch is based on a predefined threshold called an idle state threshold which acts as a boundary value for the switch decision to be made. A switch is made to power save (OS mode) as it optimizes power and creates very less compute overhead. This switch to a built-in optimizer has brought significant load reduction helping to bring down the power overhead and utilization of compute resources.

II. Proposed Methodology

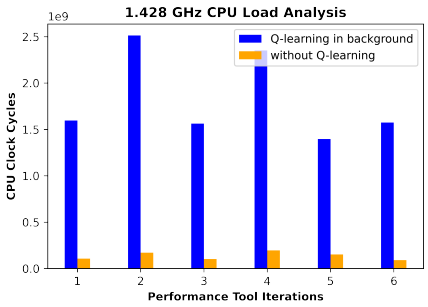
A. Proposed dynamic load based switching

One of the significant contributions of the proposed methodology is to achieve dynamic switching of modes based on the CPU load input. If we see the Figure 1 which is a plot of CPU load only created by the RL algorithm when no other active load is running. The additional compute load created by the algorithm can clearly be distinguished. This additional load by RL in the background creates power overhead and significant contributor to compute resources. This additional load problem in certain CPU states is addressed by the method of Dynamic load-based switching as shown in Algorithm 1. During the dynamic switch, the operational mode of the CPU is changed to power save. Powersave being a built-in OS mode drastically brings down the CPU load compared to Q-learning continuously running in the background.

As shown in Algorithm 1 the dynamic load-based switching happens with respect to a value called the idle state threshold. The idle state is the phase of an application when the CPU utilization is low. The idle state threshold is calculated with respect to CPU clock cycles. CPU clock cycles corresponding to low CPU utilization ($<$10%) and high CPU utilization ($>$10%) are plotted. This data can be collected by running different workloads in background and plotting the CPU clock cycles to CPU utilization. The boundary in terms of CPU clock cycles which distinguishes both these low utilization and high utilization states acts as the idle state threshold and helps us take a dynamic mode switching decision.

Algorithm 1 Dynamic load switching

```
for Monitor CPU load do
    if CPU load > idle state threshold then
        Q-learning mode
        Implement Q-table action
        Calculate payoff
        Store performance and power metrics
    else
        Activate OS based power save API
        Store performance and power metrics
```



(a)

Figure 1: CPU load comparison plots with and without Q-learning algorithm and no other active workload running operating at 1.428 GHz

B. Proposed Q-learning mode with memory management

Q-learning is an RL technique which maintains a table called Q-table across different optimization decisions (Power optimization in our use case) which can be taken. CPU clock cycles form our state space and DVFS API forms our action space. Conventional Q-learning [14] training is implemented on this state space, action space and reward function with our memory management policy.

Q-table memory management: One reason for the large size of the Q-table is multiple variables used to define the input state space. A collection of Performance Monitoring Counters (PMC) events like CPU clock cycles, cache misses etc form the state space in Q-learning. A larger Q-table makes the RL methodology not fit on edge device memory. We propose a state space optimization policy which decreases the size of the Q-table. For our state space optimization flow, a subset of the workloads is considered and run in the processor to collect PMC behaviour before running the actual Q-learning algorithm. Subset of workloads is taken from MiBenchmark suite using functions that mimic behavior of loads used in the testing of the algorithm. MiBenchmark suite has embedded multi threading in their functions. Moreover these subsets used for offline modelling have very good correlation with the online workloads. Heat maps indicating the correlation of PMC events to reward function are plotted. These plots comprise of the complete state, action space and the reward they generated based on the interaction with the specific PMC event. Red cells of the map are the negative rewarded space, green cells are the positive rewarded space and yellow is the unexplored space as illustrated in Figure 2. Heat maps with a majority of green areas are said to be converging to the target reward function and heat maps with a majority of red are not in correlation with the target reward function. The best correlating PMC event is selected using the flow below:

- The reward function of the Q-learning is modelled with the offline PMC data for test loads one PMC event mapped at a time.
- Heat maps indicating behavior of each PMC event on the reward function are plotted as shown in Figure 2.

Table I: Heat map distribution for different PMC events

PMC Event	Positive Reward (%)	Negative Reward (%)	Not Explored (%)
PMC_1	56.3	20.3	23.4
PMC_2	19.8	65.9	14.3
PMC_3	12.3	74.1	13.6
PMC_4	27.1	53.4	19.5
PMC_5	31.4	46.9	21.7
PMC_6	43.1	32.7	24.2

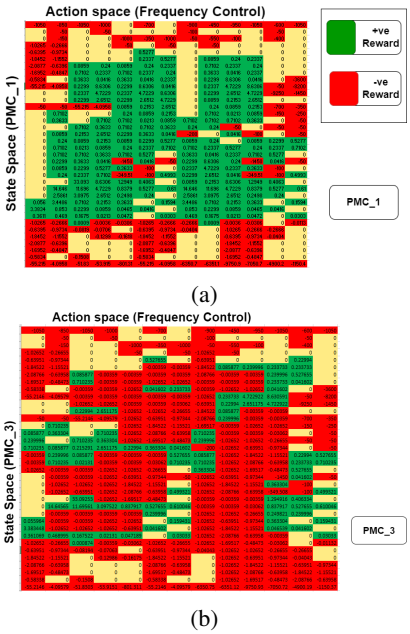


Figure 2: Heat Map of different PMC events (a) Most +ve rewarded PMC event (PMC_1) as denoted by majority green cells and (b) Most -ve rewarded PMC event (PMC_3) as denoted by majority red cells.

- Heat map of PMC event with highest positive reward is selected.

From the above procedure, we select a state space with only one variable which helps in bringing down the memory of the Q-table. From Table I we can conclude PMC_1 has highest correlation with reward. PMC_1 corresponds to CPU clock cycles in ARM A57 as per processor specs. Hence, CPU clock cycles are selected as a variable which defines the truncated state space. This method is independent of the reward function and input PMC events set.Independent of PMC event set means from any combination of PMC events one most correlated event to reward can be selected. For any arbitrary reward function and any set of PMC events mapping to one variable state space can be done using our policy.

III. Results and Analysis

The proposed methodology is validated on the A57 processor of the Jetson Tx2 board as a run time program in the embedded OS. The validation of the proposed algorithm is done by running MiBenchmarks [15] continuously in loop and average values of power and execution times recorded.

A. Reduction in Q-table size

Table. II represents the comparison of Q-table sizes for different works proposed in this domain. Q-learning methods used in HPC have a very large Q-table size per iteration greater than 100 KB per iteration as there is no strict memory constraint in HPC. The state of the art Q-learning on edge has a Q-table size of 10 KB per iteration. Our proposed methodology decreases Q-table size per iteration to 4 KB using the memory management policy. A decrease in Q-table size by 60% is achieved by our methodology compared to standalone Q-learning.

B. Distributed load scenarios (CPU utilization statistics)

The passive phase is when the CPU utilization is low ($<$10%) all other utilization states ($>$10%) are called the active phase. There is a need to generalize the amount of time an application runs in the active phase and passive phase. This is done by running all the benchmarks and observing the CPU utilization patterns. The average active phase lasted from 50%-70% of total execution time. Hence, power and performance analysis is carried on distributed load scenarios of 50% (Active phase) - 50% (Passive phase), 60% (Active phase) - 40% (Passive phase) and 70% (Active phase) - 30% (Passive phase).

C. Average Power Comparison

Figure 3a represents the average power consumption of benchmarks for different modes. Here we are considering $50\% - 50\%$ load distribution for active and passive phase. The power saving was in range of $21\% - 25.8\%$.

Figure 3b represents the average power consumption of benchmarks for different modes. Here we are considering $60\% - 40\%$ load distribution for active and passive phase. The power saving was in range of $12.3\% - 19.2\%$.

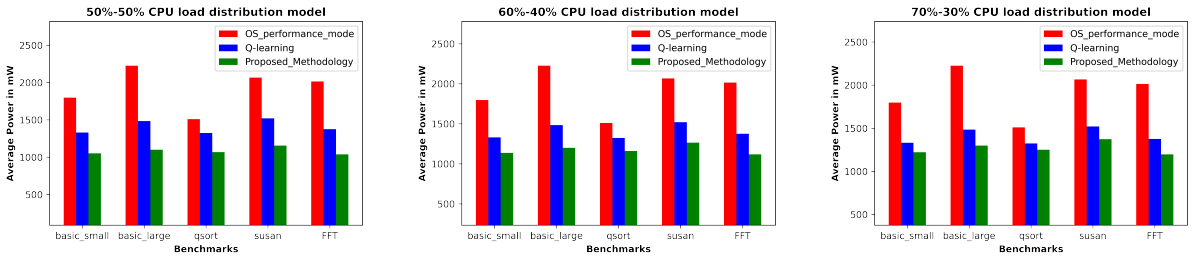
Figure 3c represents the average power consumption of benchmarks for different modes. Here we are considering 70%-30% load distribution for active and passive phase. The power saving was in range of $5.5\% - 13.1\%$. All comparisons are with respect to standalone Q-learning

D. Average Power Savings

From the results of three CPU load distribution scenarios as shown in Table. III, the average power consumption when using the proposed methodology improved in the range of $5.5\% - 25.8\%$ at the CPU level. Major contributor of power saving is powersave mode run during idle phase of any workload in place of Q-learning optimization which decreases the algorithmic load in these states bringing down the power. The average power savings for different distributions is 16.63% compared to standalone Q-learning.

E. Performance Estimation

The proposed methodology is able to bring the proposed savings in power without much degradation in the performance as shown in Table. III. This estimation is done by analyzing the execution time to run the loads. The performance degradation is in range 2% - 3.4% compared to standalone Q-learning.



(a) (b) (c)

Figure 3: Average power consumption plots of benchmarks in different Active Phase - Passive Phase CPU utilization scenarios (a) 50% (Active) - 50% (Passive) (b) 60% (Active) - 40% (Passive) and (c) 70% (Active) - 30% (Passive).

Table II: Comparison of Q-table size and optimization policy with related works

Paper	State vector size	Action Vector Size	Q-table Size	Application	Memory Management	Algorithmic load management
Yiming Wang [1]	2 + (Feedback table)	1	140 KB	HPC	✗	✗
Hao Shen [2]	3	1	120 KB	HPC	✗	✗
M Walker [14]	1	2	10 KB	Edge	✗	✗
Proposed Methodology	1	1	4 KB	Edge	✓	✓

Table III: Comparison of different modes power and performance with our proposed methodology

Benchmarks	OS_Performance		OS_Powersave		Q-learning(Edge) [14]		Proposed Methodology	
	Power(mW)	Exec_time(s)	Power(mW)	Exec_time(s)	Power(mW)	Exec_time(s)	Power(mW)	Exec_time(s)
Basic math small	1720	0.0256	1050	0.058	1310	0.03	1150	0.031
Basic math large	2200	0.0883	1060	0.17	1460	0.1	1170	0.0998
Qsort	1480	0.004	998	0.0158	1250	0.0076	1140	0.0079
Susan	1990	0.005	1030	0.0138	1475	0.0065	1250	0.0063
FFT	1800	0.0082	1003	0.019	1300	0.01	1100	0.012

IV. Conclusions

In this paper, we proposed a novel lightweight RL methodology to improve the run-time power consumption of an edge compute device. RL memory management was achieved by forming a state space with one variable. Algorithmic load management is achieved using a dynamic switch methodology which intelligently switches to either Q-learning or low power mode. The proposed methodology is validated on the Jetson Tx2 platform with ARM A57 processor for different loads. We are able to bring down Q-table size by 60% and run-time power consumption by an average of 16.63% at the CPU level compared to standalone Q-learning.

References

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

[2] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, pp. 1–32, 2013.

[3] G. Dhiman and T. S. Rosing, "System-level power management using online learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 5, pp. 676–689, 2009.

[4] J. L. Conradihoffmann and A. A. Frohlich, "Online machine learning for energy-aware multicore real-time embedded systems," *IEEE Transactions on Computers*, 2021.

[5] R. G. Kim, W. Choi, Z. Chen, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, "Imitation learning for dynamic vfi control in large-scale manycore systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 9, pp. 2458–2471, 2017.

[6] H. Shen, J. Lu, and Q. Qiu, "Learning based dvfs for simultaneous temperature, performance and energy management," in *Thirteenth International Symposium on Quality Electronic Design (ISQED)*, pp. 747–754, IEEE, 2012.

[7] W. Liu, Y. Tan, and Q. Qiu, "Enhanced q-learning algorithm for dynamic power management with performance constraint," in *DATE 2010*, pp. 602–605, IEEE, 2010.

[8] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli, "Dynamic voltage scaling and power management for portable systems," in *Proceedings of the 38th annual Design Automation Conference*, pp. 524–529, 2001.

[9] V. Devadas and H. Aydin, "On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications," *IEEE Transactions on Computers*, vol. 61, no. 1, pp. 31–44, 2010.

[10] J. Dai and Z. Liu, "Q-learning based dvfs for multi-core real-time systems," in *The International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery*, pp. 327–336, Springer, 2021.

[11] J. Hopper *et al.*, "Using the linux cpufreq subsystem for energy management," *IBM blueprints*, 2009.

[12] Z. Mwaikambo, A. Raj, R. Russell, J. Schopp, and S. Vaddagiri, "Linux kernel hotplug cpu support," in *Linux Symposium*, vol. 2, p. 2004, 2004.

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

[14] A. Das, M. J. Walker, A. Hansson, B. M. Al-Hashimi, and G. V. Merrett, "Hardware-software interaction for run-time power optimization: A case study of embedded linux on multicore smartphones," in *ISLPED*, pp. 165–170, 2015.

[15] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*, pp. 3–14, IEEE, 2001.