Q-learning Based Backup for Energy Harvesting Powered Embedded Systems

Wei Fan, Yujie Zhang, Weining Song, Mengying Zhao*, Zhaoyan Shen, Zhiping Jia
School of Computer Science and Technology, Shandong University, Qingdao, China

***Abstract*—Non-volatile processors (NVPs) are used in energy harvesting powered embedded systems to preserve data across interruptions. In NVP systems, volatile data are backed up to non-volatile memory upon power failures and resumed after power comes back. Traditionally, backup is triggered immediately when energy warning occurs. However, it is also possible to more aggressively utilize the residual energy for program execution to improve forward progress. In this work, we propose a Q-learning based backup strategy to achieve maximal forward progress in energy harvesting powered intermittent embedded systems. The experimental results show an average of 307.4% and 43.4% improved forward progress compared with traditional instant backup and the most related work, respectively.**

***Index Terms*—Energy harvesting system, Non-volatile processor, Q-learning, Forward progress**

I. Introduction

With development of technology, processors tend to be smaller and more energy efficient, which makes them qualified to be embedded everywhere, such as wearable devices including shoes, watches and glasses, as well as Internet of Thing edge devices deployed in forests and mountains. Energy harvesting is more suitable for powering these embedded devices compared with batteries which have limitations such as large size and weight, high price, and huge maintenance overhead [1]. Energy harvesting systems can gain energy from surrounding environments, and the harvested energy will be converted into electronic power to charge capacitor or power embedded devices directly. Typical energy sources include solar, wind, RF and vibration, which are green, but unstable [2]. Traditional CMOS based processors may be interrupted frequently under unstable power supply since volatile states will be lost after power off. When power resumes, the system needs to reboot and execute program from beginning, causing severe performance degradation. What is worse, large programs cannot be completed with frequent power offs since immediate results cannot be saved.

A processor attached with non-volatile memory (NVM), called non-volatile processor (NVP), has been proposed to store volatile data when power failure occurs [3]. After power resumes, these stored data will be copied back into processor, which enables the system to continue executing programs from where being interrupted. Fig. 1 demonstrates a fabricated NVP architecture, where the non-volatile flip-flop (NVFF) logic is designed to build non-volatile register files [3]. In a system with more complicated memory hierarchy, all volatile memory levels, such as register file, cache and main memory, need backup support to achieve accumulative program execution [4]–[6].

*Corresponding author: zhaomengying@sdu.edu.cn

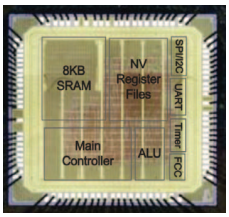

Fig. 1. A typical architecture of NVP [3].

Forward progress can be used to evaluate the performance of intermittent NVP systems. It is defined as the progress of program execution with existence of power failures. There are researches focusing on forward progress improvement so that the program can be finished at an earlier time even with interruptions [7]–[10].

In this paper, we introduce Q-learning to NVP system to maximize forward progress by dynamically determining when to back up. Instead of comprehensive system behavior tracking and backup data analysis, it leverages the learning procedure to figure out how to make backup decision based on current system status. To the best of our knowledge, it is the first work to use Q-learning for backup guidance in NVP systems. Specifically, we make the following contributions in this paper.

- Propose to introduce Q-learning to guide runtime backup for forward progress improvement in NVP systems, which can make backup decision at appropriate time dynamically;
- Design a type of reward function and a back-track update method to adapt Q-learning algorithm to the specific backup decision-making problem;
- Develop a gem5 based simulator to evaluate the proposed approach and compare with the most related work.

The rest of this paper is organized as follows. Section II summarizes related work. Section III first gives the problem overview and then presents the proposed Q-learning based backup method in detail. Section IV shows experimental results and discussions. Section V concludes this paper.

II. Related Work

This section summarizes related work on forward progress improvement in NVP systems and also the application of Q-learning in energy harvesting systems.

A. Forward Progress Improvement in NVP Systems

The essential difference between NVP system and traditional VP system is the backup/resumption procedure. In NVP systems, after receiving an energy warning, the energy stored in capacitor will be used to support system backup. Thus the volume of the capacitor is typically designed no less

than the worst-case energy consumption for backing up all volatile data. There are researches working on reducing data size to back up to improve system energy efficiency, such as selective backup [7], stack size aware backup [9], and compression based backup [11]. As for the backup timing, the most straightforward way is to trigger backup once receiving the energy warning, which is called instant backup. Since the capacitor volume is conservatively defined, it is potential to aggressively use the energy in it to execute the program a little bit more before conducting backup, leading to improved forward progress. Ransford et al. [12] develop a software system called Mementos, which enables long-running computations to span power loss events. At compile time, Mementos inserts trigger points which are calls to a Mementos library function that estimates available energy, at control points such as loop-latch and function return in the program. At runtime, Mementos detects the capacitor voltage and triggers the checkpoint. Li et al. [8] employ offline cache behavior study to analyze content to backup at each position in order to guide runtime backup at the furthest possible location. These researches improve forward progress with the assistance of offline program analysis and modification. Ma et al. [10] propose a runtime backup strategy to dynamically determine the number of instructions to execute after receiving energy warning. The decision is made based on historical behaviors. In this work, we also target an online backup decision maker to achieve maximal forward progress in NVP systems.

B. Q-learning in Energy Harvesting Systems

Reinforcement learning (RL) model consists of a decision-making agent, a set of states S, a set of actions A for each state s_t, denoted $A(s_t)$, and designed rewards R, with which agent learns to make better decisions by interacting with the environment, as illustrated in Fig. 2(a). Agent performs an action a_t ($a_t \in A(s_t)$) at state s_t, and it will move to next state s_{t+1}. The next state and its reward $R_{t+1}(a_t,s_t)$ are provided to the agent by environment. The general goal of agent is to maximize its total cumulative reward by learning which action is appropriate for a particular state.

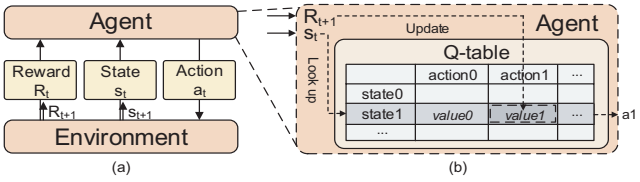

Fig. 2. (a) The framework of RL. (b) The agent of Q-learning.

Q-learning is a popular reinforcement learning method, which is a formal framework that models sequential decisions. In Q-learning, the agent maximizes total reward by evaluating the quality Q of state-action pair and storing historical Q values into a Q-table as guidance of next action selection. As shown in Fig. 2(b), the agent can look up Q-table according to state s_t, and select action $a1$ whose Q value is larger. The new Q value calculated with R_{t+1} given by environment is updated into corresponding location of Q-table later for future reference.

Q-learning based strategies have been developed in energy harvesting systems for decision-making under uncertainties. In environment monitoring systems, Prauzek et al. [13] propose a Q-learning based energy management to dynamically change sample frequency of sensors, which is influenced by battery level of sensors, input power level and data buffer level. In wireless body area networks (WBANs), Kazemi et al. [14] develop a power control approach to save energy consumption per bit substantially. To extend node's lifespan in WBANs, Rioual et al. [15] expect to reduce node's consumption when the harvested energy is less. So, they employ Q-learning algorithm in determining node's work frequency with monitoring residual energy in battery and energy gained from environment. Although Q-learning is used in energy harvesting systems, there is no work about system backups. In this work, we aim to explore how to introduce Q-learning to NVP system for backup guidance, so that the forward progress can be optimized.

III. Q-learning Based Backup Method

In this section, we will first introduce the framework of the proposed Q-learning based backup method which aims to improve forward progress in self-powered NVP system. Then the technical details and algorithm are described. optimized.

A. The Framework of Q-learning Based Backup

Fig. 3 demonstrates the architecture of the target NVP system. Energy harvested from environment is used to charge the capacitor as the system power supply after voltage conversion. Registers in the system, which are in small size and frequently updated, are backed up with NVFF [3], where all registers can be backed up simultaneously from the volatile flip-flop to non-volatile portion after receiving energy warnings. Cache in the system is volatile and uses on-demand selective backup, i.e., only dirty lines need to be backed up to non-volatile memory. For simplicity, we assume the main memory is non-volatile and does not need backup. Actually, the proposed technique is easy to be extended to a different architecture with volatile main memory, too.

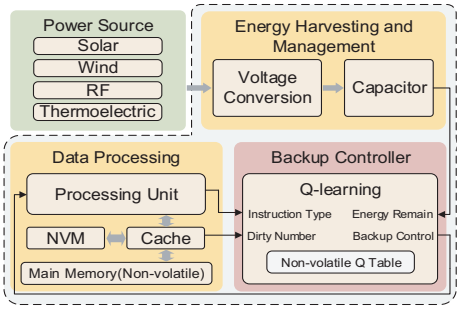

Fig. 3. The framework of Q-learning based backup method.

A Q-learning based backup method serves as the backup controller to determine the exact timing to back up after receiving energy warnings. Using specific information provided by capacitor and processing unit, the backup controller works according to designed reward functions and Q-table, in order to appropriately send backup signals to the data processing unit.

B. Q-learning for Backup Decision-making

The key to apply Q-learning to NVP system is to appropriately define agent behavior, state, action and reward in the learning framework, so that the guided backup after learning can achieve maximal forward progress.

Fig. 4 shows the working process of NVP system with Q-learning based backup controller. When the energy is sufficient, the system works without involvement of the Q-learning based controller (Fig. 4(a)). When energy warning occurs, the controller determines the backup location and triggers backup (Fig. 4(b)). When enough energy is harvested again, data are resumed for program continuation (Fig. 4(c)). The backup controller learns based on the following steps:

1) The environment provides current states s_t to agent;
2) The agent chooses one action based on Q values;
3) The environment gives feedback rewards or punishments to the agent;
4) The agent updates Q value of the chosen action in Q-table.

After a period of time with iteratively learning from the environment, the agent is able to build an effective Q-table to guide the backup decision. In the following, we will discuss the design of each component in detail.

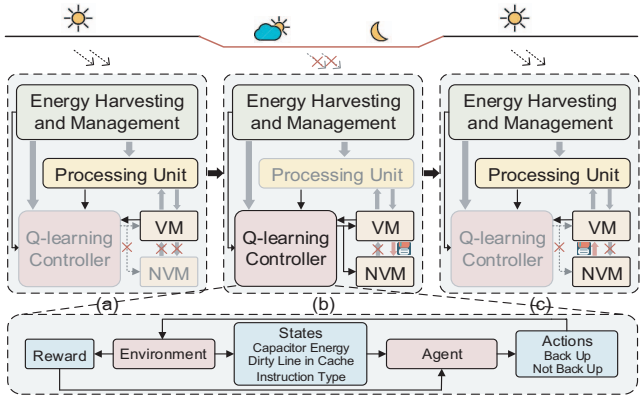

Fig. 4. Working process of NVP system with Q-learning based controller.

1) **Agent:** The agent is the brain of the whole system. In a particular state, agent selects action according to agent's policy and then it will receive rewards or punishments. The policy is a sequence of actions taken by agent, and the best policy is such a sequence that can maximize total rewards. To find the best policy, many trial-and-error steps have to be performed.

The quality Q of state-action pair is calculated by the formula [13]:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \cdot (R_{t+1} + \gamma \cdot max_{(\forall a \in A)}[Q_t(s_{t+1}, a_t)] - Q_t(s_t, a_t)) \quad (1)$$

In the above formula, $Q_{t+1}(s_t, a_t)$ represents the new quality after performing a_t in state s_t, and $Q_t(s_t, a_t)$ is the old quality value. The learning rate α determines how strongly the previous training results influence current selection. If α is set as 0, Q-learning would not teach the agent the latest information. The discount factor γ represents the future effect. A factor of 0 would make the agent only consider current rewards.

Using Formula (1), the agent can learn optimal policy to take action in a particular state. Usually, the agent can simply select the action with the maximal Q value. However, in order to avoid the local optimal solution, we adopt the ϵ-greedy strategy [13], shown as follows:

$$a_{t+1} = \begin{cases} random\ action\ from\ A(s_{t+1}) & ,if\ \zeta < \epsilon \\ argmax(Q(s_{t+1}, a)) & ,otherwise \end{cases} \quad (2)$$

where ζ is a random number ranging from 0 to 1, and ϵ is a defined random rate. If ζ is less than ϵ, the agent will randomly select an action at this state, otherwise, taking the action with the maximal Q value. The ϵ-greedy strategy can explore hidden state-action pairs, even if these pairs have lower Q values.

2) **Actions and States:** As previously described, the agent is in charge of selecting one action for current state according to the maintained Q-table. In NVP systems, it determines whether to back up or not in the current step. Thus the action set in this problem can be defined as "back up" and "not back up".

The states need to be carefully designed by considering factors affecting backup decision. The system receives an energy warning when the energy in capacitor drops to a threshold, which is defined as the required energy for backup in worst case. Since the content to back up at each position is different, the system has potential to more aggressively use energy in capacitor to execute more instructions before conducting backup. In order to guarantee successful backup, as long as the energy in capacitor is obviously sufficient for backup, the system can continue program execution. Otherwise, when the energy in capacitor drops close to that needed for system backup, it should decide to back up. Inspired by this, the state information should contain energy available in capacitor and also energy required for backup. Energy in capacitor can be directly obtained from the energy harvesting module. But the required energy for backup is not easy to derive. In addition to current number of dirty lines in cache, the type of the following instruction may also affects content to back up. For example, computation instruction only changes register file while memory-access instructions may increase or decrease the number of dirty lines in cache. Based on this consideration, we add two elements into the states to indicate the energy required for backup, i.e., the number of dirty lines in cache, and the following instruction type. Here register file is not listed as a consideration for state design since they are always fully backed up with NVFF and the required backup energy is constant. To sum up, the state is defined as a triple pair ⟨capacitor energy, # of dirty lines in cache, instruction type⟩. When agent receives states s_t from environment, it will select the preferred action from the action list corresponding to state s_t in the Q-table, by referring to Formula (2). After executing the chosen action, agent enters next state s_{t+1} and obtains

feedback of reward R_{t+1} from environment. In this work, in addition to reward definition, we also introduce punishment for agent learning.

3) **Rewards and Punishments:** Rewards are defined to tell the agent how good the last decision on action is. In the target problem, a big reward should be given if successful backup can be guaranteed and large forward progress can be achieved.

We design a type of reward function related with forward progress and remaining energy, as shown in Formula (3). E_{remain} represents the energy remained in capacitor after taking action a_t, and *FP* is the forward progress from previous checkpoint to current location, where p and q are positive numbers.

$$R_{t+1} = \begin{cases} p \cdot E_{remain} & , \textit{if } a_t \textit{ is not back up} \\ -p \cdot E_{remain} + q \cdot FP, & \textit{else if } a_t \textit{ is back up} \end{cases} \quad (3)$$

When E_{remain} is large, the action of "not back up" has a higher reward, indicating a preferred action of continuing program execution. As the energy is consumed and *FP* increases, the reward of a_t being "back up" will exceed that of "not back up". By appropriately setting values of p and q, the agent is able to figure out the best position for backup.

During exploration of Q-learning, the agent may choose *bad* actions. For instance, energy remained in capacitor is not enough to support backup, causing backup failures. Or, it always chooses "not back up" in a too aggressive way, leading to power failures during instruction execution even without triggering backup yet. As a result, the system needs to roll back to the last checkpoint after power resumes. In this situation, the agent should receive a punishment in order to decide an earlier backup next time to avoid rollbacks. Here, punishment is defined as a negative reward. Besides, we adopt a back-track method to also give punishments to previous *bad* decisions which contribute to current failure. To do that, we record several previous states using *collection*, and when failures occur, punishments will be given to all states recorded in *collection*.

C. Algorithm of Q-learning Based Backup

In this subsection, we will describe the algorithm of our Q-learning based backup method formally in Algorithm 1. After initializing Q-table to a matrix of zeros and ϵ to 1 that is the maximal value (Line 1-3), c is set to empty (Line 4) which is a *collection* used to record old states for back-track. Then states of environment (i.e. energy remained in capacitor, number of dirty lines in cache and instruction type) are served as current states (s_{curr}) (Line 5). Once energy warning occurs, the Q-learning algorithm will be triggered. First, the action is selected by ϵ-greedy strategy (Line 9-14), as shown in Formula (2). ϵ is initialized to 1 at beginning, making the agent select random actions in the preliminary exploration phase. As the Q-learning algorithm proceeds, ϵ is decreased by δ (Line 28), making agent select more exploitive actions. Then, the agent takes the selected action a (Line 15). Later the environment changes its states and sends the updated states (s_{next}) to the agent (Line 16). There is a specific state that is *fail*. Once

Algorithm 1 Algorithm of Q-learning Based Backup Method

```
1:  Initialize Q_table as an empty set;
2:  Initialize s_curr, s_next, a;
3:  ε ← 1;
4:  c ← ∅;
5:  s_curr ← monitor environment;
6:  c ← c ∪ s_curr;
7:  while Q_table not converge do
8:    if energy emergency occurs then
9:      Random ζ ;
10:     if ζ < ε then
11:       a ← random action from A(s_curr);
12:     else
13:       a ← argmax(Q_table[s_curr, a']);
14:     end if
15:     Take action a;
16:     s_next ← monitor environment;
17:     c ← c ∪ s_next;
18:     if s_next is fail then
19:       for each state s' in c do
20:         Q_table[s', a] ← punishments;
21:       end for
22:       Roll back to last checkpoint;
23:     else
24:       r ← reward(s_next, a);
25:       q' ← r + γ · max(Q_table[s_next, a']);
26:       Q_table[s_curr, a] += α · (q' − Q_table[s_curr, a]);
27:     end if
28:     ε ← max(ε − δ, ε_min);
29:     c ← c \ s_curr;
30:     s_curr ← s_next;
31:   end if
32: end while
```

Q-learning algorithm detects s_{next} is *backup fail* or *execution fail*, corresponding value in Q-table is changed by punishment and program will roll back to last checkpoint (Line 18-22). If s_{next} is not *fail*, the corresponding Q value is updated according to Formula (1) (Line 24-26). Finally, the new state is saved into current state (Line 30). This procedure is executed iteratively until the values in Q-table are converged.

For learning rate α, we decrease it slowly according to Formula (4) as the algorithm executes, making the proposed method more stable and converge faster. In Formula (4), N is the number of iterations this algorithm has already been executed and C is a positive constant number related to program size. To balance exploration space between the rate of convergence, M decreases more and more slowly.

$$\alpha \ = \alpha - M \cdot \lfloor \frac{N}{C} \rfloor \quad (4)$$

After learning in this way, the agent is able to generate an effective and efficient Q-table, which can be used in future backup to achieve improved forward progress. Note that the Q-table can be further updated when necessary during following usage.

IV. Experiment and Results

This section will first introduce experiment setup, and then experiment results and discussions are shown.

A. Experiment Setup

Gem5 [16] is a highly configurable CPU architecture simulator. The proposed Q-learning based backup method is implemented in gem5 simulator. Table I describes the settings of gem5 simulator in detail, and Table II presents the access latency and energy consumption of different storage that are attained by NVSim [17]. The parameters used in the proposed Q-learning based strategy are listed in Table III, whose values are determined empirically.

TABLE I
System Settings

Component	Configuration
Processor	1 core, 72MHz, NVFF based register file
L1 I/D Cache	16kB SRAM, 8-way associative, 64B cache line, write back
Main Memory	512MB STT-RAM
Non-Volatile Memory	16kB STT-RAM for cache backup

TABLE II
Access Latency and Energy Consumption

	Cache (SRAM)	NVM (STT-RAM)
Write Latency (cycle)	2	11
Write Energy (pJ per line)	81	527
Read Latency (cycle)	2	3
Read Energy (pJ per line)	81	122

TABLE III
The Value of Parameters

Parameters	Discount factor (γ)	Learning rate (α)	E_{remain} factor (p)	FP factor (q)
Value	0.5	0.5	0.5	0.2
Parameters	ϵ decrement (δ)	ϵ min (ϵ_{min})	Decline factor (M)	α factor (C)
Value	0.0004	0.1	0.05-0.01	100000

Benchmarks are from Mibench [18], which are representative applications in embedded systems. We adopt a 12-hour solar trace [19], which is measured from real world, as the power supply of the NVP system. When backup is completed, the NVP system enters sleep mode and waits for the capacitor being fully charged for resumption.

B. Experiment Results

We compare the proposed Q-learning based backup scheme with instant backup method and the most related strategy called ALD proposed in [10]. The instant backup method assumes all cache blocks are dirty and always reserves corresponding energy for backup. Once the energy in capacitor drops to this value, it triggers backup instantly. Differently, ALD tries to learn how many instructions can be executed using residual energy after backup. Then the number of executed instructions will be recorded into a table indexed by input power level. After the learning procedure, when energy warning occurs, the system looks up the table according to current input power level, and figures out the number of instructions to execute before backup. In this manner, it makes decision according to historical experience to improve forward progress.

Fig. 5 shows the forward progress of the proposed Q-learning based backup method, ALD strategy and instant backup. All data are normalized to those of ALD. The forward progress is defined as the number of instructions having been executed supported by the whole power trace. For Q-learning and ALD, learning is first conducted until the values in Q-table and look-up table are converged, after which the forward progress is measured. It shows that forward progress improvements the proposed Q-learning based method can achieve over ALD and instant backup are 43.4% and 307.4% respectively on average.

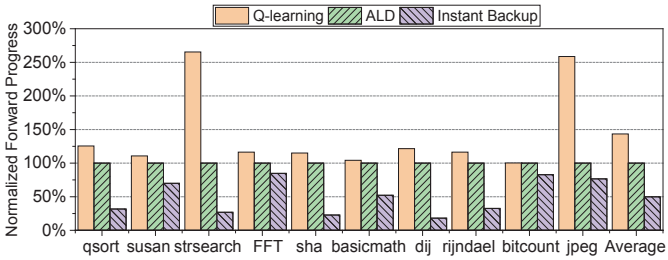

Fig. 5. Forward progress improvement.

C. Discussions

It can be observed from Fig. 5 that all benchmarks with Q-learning algorithm have obvious forward progress improvement compared with instant backup method. This is because instant backup triggers more backups since it immediately triggers backup once receiving energy warnings. Thus it takes less energy for program execution.

Since ALD tries to aggressively execute instructions after energy emergency occurs, it performs better than instant backup. However, the performance of ALD strongly depends on the distribution of different types of instructions in the program. For example, memory-access instructions usually consume more energy than calculation instructions. If the instructions during learning procedure have a high ratio of calculation instructions, the record in table tends to execute a large number of instructions before backup. However, there may be more memory-access instructions in real case. When the same input power level appears afterwards, if it takes the suggestion looked up from the table, it may lead to backup failures and need to roll back to the last checkpoint after energy resumes. In contrast, if the learned number is too conservative due to a large portion of memory-access instructions, ALD may only have limited forward progress without fully utilizing the remaining energy. Thus the performance of ALD highly depends on the distribution of calculation/memory-access instructions in the program. In *bitcount* and *basicmath* with comparatively uniform instruction distribution, ALD achieves close forward progress to the proposed Q-learning based strategy. However, in *strsearch* and *jpeg* with nonuniform distribution, ALD has more backup failures and rollbacks, as shown in Table IV, leading to degraded forward progress.

TABLE IV
COMPARISON ON NUMBER OF BACKUP AND ROLLBACK

Benchmark \ Method	Q-learning		ALD		Instant Backup	
	#backups	#rollbacks	#backups	#rollbacks	#backups	#rollbacks
qsort	2	0	6	1	55	0
susan	3	0	3	1	28	0
strsearch	9	0	11	5	40	0
FFT	3	0	3	1	31	0
sha	4	0	11	0	74	0
basicmath	5	0	6	0	50	0
dij	14	0	17	1	69	0
rijndael	13	0	11	3	132	0
bitcount	2	0	2	1	16	0
jpeg	17	0	11	6	42	0

The advantage of the proposed Q-learning based backup comes from more precise information learned from capacitor and system status such as dirty lines in cache and instruction types. Besides, the agent is punished when backup or execution fails, and thus there is no rollback after learning. As Table IV shows, Q-learning based backup has no rollbacks and less number of backups than ALD. It also indicates a higher energy utilization in Q-learning based backup.

V. CONCLUSION

In this work, we propose a Q-learning based backup method to determine when to backup dynamically with the aim of maximizing forward progress in NVP systems. In this approach, energy remained in capacitor, the amount of data need backup and instruction type are treated as states, and backup or not are actions determined by Q-learning algorithm. The system can make appropriate decision in particular states with guidance of reward functions and punishments. Compared with instant backup strategy and the most related work ALD, the proposed approach can improve forward progress by 307.4% and 43.4%, respectively.

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation for Young Scientists of China (NSFC) under Grant No.61902218, and the Young Scholars Program of Shandong University (YSPSDU) under Grant No.2018WLJH62.

REFERENCES

[1] M. Hicks, "Clank: Architectural support for intermittent computation," in *Proceedings of IEEE International Symposium on Computer Architecture (ISCA)*, Toronto, Canada, 2017, pp. 228–240.

[2] Y. Liu, J. Yue, H. Li, Q. Zhao, M. Zhao, C. J. Xue, G. Sun, M.-F. Chang, and H. Yang, "Data backup optimization for nonvolatile sram in energy harvesting sensor nodes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 10, pp. 1660–1673, 2017.

[3] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, M.-F. Chiang, Y. Yan, B. Sai, and H. Yang, "A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops," in *Proceedings of IEEE European Solid-State Circuits Conference (ESSCIRC)*, Bordeaux, France, 2012, pp. 149–152.

[4] M. Xie, M. Zhao, C. Pan, H. Li, Y. Liu, Y. Zhang, C. J. Xue, and J. Hu, "Checkpoint aware hybrid cache architecture for nv processor in energy harvesting powered systems," in *Proceedings of ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, USA, 2016, p. 22.

[5] K. Ma, Y. Zheng, S. Li, K. Swaminathan, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan, "Architecture exploration for ambient energy harvesting nonvolatile processors," in *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA)*, San Francisco, USA, 2015, pp. 526–537.

[6] H. Li, Y. Liu, Q. Zhao, Y. Gu, X. Sheng, G. Sun, C. Zhang, M.-F. Chang, R. Luo, and H. Yang, "An energy efficient backup scheme with low inrush current for nonvolatile sram in energy harvesting sensor nodes," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2015, pp. 7–12.

[7] K. Ma, X. Li, S. Li, Y. Liu, J. J. Sampson, Y. Xie, and V. Narayanan, "Nonvolatile processor architecture exploration for energy-harvesting applications," *IEEE Micro*, vol. 35, no. 5, pp. 32–40, 2015.

[8] J. Li, M. Zhao, L. Ju, C. J. Xue, and Z. Jia, "Maximizing forward progress with cache-aware backup for self-powered non-volatile processors," in *Proceedings of IEEE Design Automation Conference (DAC)*, Austin, USA, 2017, pp. 1–6.

[9] M. Zhao, C. Fu, Z. Li, Q. Li, M. Xie, Y. Liu, J. Hu, Z. Jia, and C. J. Xue, "Stack-size sensitive on-chip memory backup for self-powered nonvolatile processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 11, pp. 1804–1816, 2017.

[10] K. Ma, X. Li, H. Liu, X. Sheng, Y. Wang, K. Swaminathan, Y. Liu, Y. Xie, J. Sampson, and V. Narayanan, "Dynamic power and energy management for energy harvesting nonvolatile processor systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 4, p. 107, 2017.

[11] Y. Wang, Y. Liu, Y. Liu, D. Zhang, S. Li, B. Sai, M.-F. Chiang, and H. Yang, "A compression-based area-efficient recovery architecture for nonvolatile processors," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, 2012, pp. 1519–1524.

[12] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on rfid-scale devices," in *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Newport Beach, USA, 2011, pp. 159–170.

[13] M. Prauzek, N. R. Mourcet, J. Hlavica, and P. Musilek, "Q-learning algorithm for energy management in solar powered embedded monitoring systems," in *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, Rio de Janeiro, Brazil, 2018, pp. 1–7.

[14] R. Kazemi, R. Vesilo, E. Dutkiewicz, and R. Liu, "Dynamic power control in wireless body area networks using reinforcement learning with approximation," in *Proceedings of IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Toronto, Canada, 2011, pp. 2203–2208.

[15] Y. Rioual, Y. Le Moullec, J. Laurent, M. I. Khan, and J.-P. Diguet, "Reward function evaluation in a reinforcement learning approach for energy management," in *Proceedings of IEEE Baltic Electronics Conference (BEC)*, Tallinn, Estonia, 2018, pp. 1–4.

[16] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News (CAN)*, vol. 39, no. 2, pp. 1–7, 2011.

[17] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 31, no. 7, pp. 994–1007, 2012.

[18] Mibench: http://www.eecs.umich.edu/mibench/.

[19] Y. Liu, Z. Li, H. Li, Y. Wang, X. Li, K. Ma, S. Li, M.-F. Chang, S. John, Y. Xie *et al.*, "Ambient energy harvesting nonvolatile processors: from circuit to system," in *Proceedings of IEEE Design Automation Conference (DAC)*, San Francisco, USA, 2015, p. 150.