Learning-Based Power Management for Multi-Core Processors via Idle Period Manipulation

Rong Ye† and Qiang Xu†‡
†CUhk REliable Computing Laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
‡Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
Email: {rye,qxu}@cse.cuhk.edu.hk

ABSTRACT

Learning-based dynamic power management (DPM) techniques, being able to adapt to varying system conditions and workloads, have attracted lots of research attention recently. To the best of our knowledge, however, none of the existing learning-based DPM solutions are dedicated to power reduction in multi-core processors, although they can be utilized by treating each processor core as a standalone entity and conducting DPM for them separately. In this work, by including task allocation into our learning-based DPM framework for multi-core processors, we are able to manipulate idle periods on processor cores to achieve a better tradeoff between power consumption and system performance. Experimental results show that the proposed solution significantly outperforms existing DPM techniques.

I. Introduction

Dynamic power management (DPM) for electronic systems, which trades off performance for power savings in a controlled fashion, is one of the most successful techniques used for energy-efficient computing [1]. To be specific, by taking system workloads into account, DPM reduces power dissipation via selectively shutting down (or lowering the performance of) inactive system components. For example, a microprocessor can be put in sleep mode for power reduction when it is idle for some time and it is waken up when new tasks arrive.

State transitions in DPM, however, involve non-trivial performance penalty and power cost, and an eager power management policy that turns off system components as soon as they are idle may even increase the system power dissipation and degrades its performance at the same time. Consequently, how to optimize the DPM policy, the procedure that takes decision on the state of the system components, is a rather complex constrained optimization problem, especially considering the fact that a component may have multiple operational modes with different power benefits and transition costs[1].

Because it is very difficult, if not impossible, to choose the opportune moment to turn off inactive components without knowing the actual workloads of the system in advance, the key issue in DPM policy optimization is how to efficiently utilize/predict the idle periods of system components to obtain power saving without much performance degradation. Earlier work in this domain tries to characterize the system workloads first and then derive an optimal policy for the system (e.g., [2]). Stochastic modeling for workloads such as Markov decision process were also used for complicated systems (e.g., [1, 3]). The effectiveness of the above techniques, however, relies heavily on the accuracy of their workload models, which are not guaranteed during the off-line optimization process. Recently, several learning-based DPM techniques have been presented in the literature (e.g., [4–6, 12]). By online learning the workload characteristics and adjusting DPM policy on-the-fly, learning-based solutions can adapt to varying system conditions and workloads and hence are potentially able to achieve better power/performance tradeoff than those off-line solutions.

Today, multi-core processors are widely used in electronic systems. Since the system behavior gets more complicated as the number of processor cores increases, building an effective off-line DPM solution may involve long trial-and-error iterations for workload modeling and learning-based solutions seem to be a natural choice. To the best of our knowledge, however, none of the existing learning-based DPM solutions are dedicated to power reduction in multi-core processors, although they can be utilized by treating each processor core as a standalone entity and conducting DPM for them separately.

In multi-core processors, global power management solutions can outperform those solutions that manage power per-core locally. This is because, given the same workloads, various task allocation strategies may lead to significantly different idle periods on each processor core, and hence have a significant impact on the efficiency of DPM policies. From a different perspective, even though the workloads are still unknown, the idle periods on processor cores become partially controllable in multi-core processors. Motivated by the above, we develop a novel learning-based DPM framework for multi-core processors that judiciously allocate tasks on processor cores to achieve a better tradeoff between power consumption and system performance. To be specific, we use Q-learning, a kind of reinforcement learning technique, to learn the system behavior and determine proper processor power state transitions. As the solution space increases exponentially with the increase of processor cores, we use neural network in our learning framework to speed up the training process. Experimental results show that our proposed power manager for multi-core processors significantly outperforms existing DPM techniques.

The remainder of this paper is organized as follows. In Section II, we survey related work in this area and motivate our work. The proposed learning-based DPM framework and the corresponding learning algorithms are then detailed in Section III and Section IV, respectively. Next, Section V presents our experimental results. Finally, Section VI concludes this paper.

II. Related Work and Motivation

A. Related Work

There are numerous related works in dynamic power management in the literature. In this work, we focus on how to conduct effective power state transitions. From this aspect, generally speaking, existing DPM policies can be classified into two categories [4]: heuristic policies and stochastic policies. Time-out policy [7] is one of the most widely-used heuristic policies, which simply turns off a component when the duration time, for which the component has been in idle period, exceeds a pre-defined time interval. Time-out policies are simple and robust, but they may be too fast or too slow to react. Stochastic policies, on the other hand, model system state changes and request arrivals as stochastic processes. Markov decision process [1] and Semi-Markov decision process [3] are often adopted to derive an optimal DPM policy according to these models.

[1] For example, a processor in deep sleep state has lower power consumption but requires more transition time and transition power when waken up, compared to that in light sleep state.

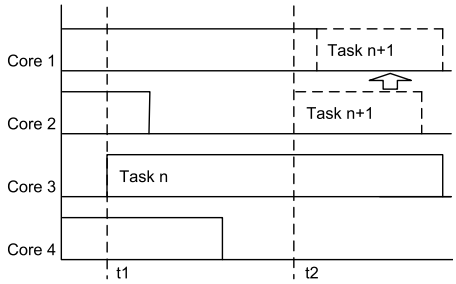

Fig. 1. Motivational Example

In the above works, DPM policies are determined at design stage and they may not work well with varying workload characteristics and environment conditions. Learning-based DPM solutions are thus attractive since they are able to adapt to varying system conditions and workloads. Srivastava *et al.* [8] explored a shutdown mechanism to predict the length of idle time based on real-life traces and recent computation history. In [2], Hwang *et al.* predicted the current idle period length using exponential average approach based on previous idle periods. In [6], Steinbach proposed reinforcement learning-based DPM policy to perform mid-level power management in wireless network cards. Theocharous *et al.* [9] considered user annoyance as a performance constraint and presented a user-based adaptive power management technique. In [4], Dhiman and Rosing proposed to dynamically select the best DPM policy from a set of candidate policies. In [12], Tan *et al.* presented an approach for system-level power management in a partially observable environment, based on model-free constrained reinforcement learning.

There are also some recent works that consider DPM in multi-core processors, which can be categorized into per-core approach [5,10,11] and chip-wide approach [13, 14]. In [10], Canturk *et al.* proposed an approach to set the power mode of each core to meet a power budget. Jung *et al.* [5, 11] presented a supervised learning-based DPM framework for multi-core processors. Their approach, however, determines power management actions for each core based on their individual workload prediction and hence is not a "true" multi-core power management scheme. In [13], Sebastian *et al.* utilized a control theory based controller to apply DVFS technique, but the task-to-core allocation is fixed in their approach. In [14], Mohammad *et al.* proposed a hierarchical DPM framework under given throughput constraint, which employs core consolidation, coarse-grained DVFS and task allocation at the CMP level and fine-grained DVFS based on closed-loop feedback control at the individual core level. This work required to obtain task characteristic a priori for task allocation.

B. Motivation

As discussed earlier, in multi-core processors, we have the flexibility to assign a task to any processor core and hence the idle periods on processor cores become partially controllable, which can be exploited for power savings. Note that, for the sake of simplicity, we assume that each task is executed on only one core and there is no dependency between tasks. In addition, we mainly consider dynamic power consumption for task execution in this work.

Fig. 1 presents the motivational example for our work. In this 4-core processor, when $task_{n+1}$ arrives, allocating it to different cores for processing may lead to very different results. Suppose the task is assigned to *core* 2. Since this core has been idle for some time, it might be in sleep mode at this time point, and we have to wake it up to process this task, causing extra power dissipation and performance penalty. If, however, the task is assigned to *core* 1, we are able to save the above cost without incurring much performance penalty since it is about to finish the task assigned to it earlier. Ideally, if we can assign a new task to a processor core that has just finished its earlier-assigned task at that time point, we do not need to suffer from any cost.

Motivated by the above, we develop a novel learning-based DPM framework for multi-core processors that judiciously allocate tasks on processor cores to achieve a better tradeoff between power dissipation and system performance, as discussed in the following sections.

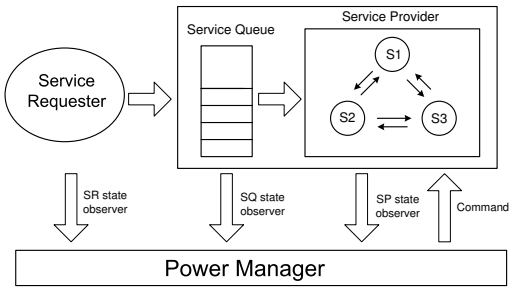

Fig. 2. Abstract Structure of DPM System Model

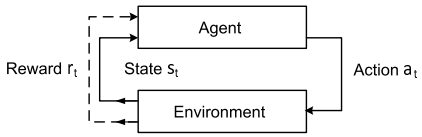

Fig. 3. Q-learning Cycle [15]

III. System Framework

A power-manageable system can be modeled as shown in Fig. 2 [1, 12], which includes three components: service requester (*SR*), service queue (*SQ*) and service provider (*SP*). SR issues requests as the event source, while SP processes requests with different power modes. SQ buffers requests that cannot be processed at once, if SP is too busy. The power manager observes system states (consisting of SR, SQ and SP states), and controls the behavior of SP, to achieve power savings at certain performance penalty. Based on the above, we setup our Q-learning model for DPM problem in multi-core processors in this section.

A. Background on Q-Learning

Q-learning, as one of the prevalent reinforcement learning algorithms, has been applied in many scientific and engineering fields. Since it is also used in our proposed DPM solution, we briefly introduce it in the following.

The basic idea of Q-learning [6, 15] is to decide on what action to take based on current system state information in order to maximize the expected *reward* in the future by mapping states to actions. In standard Q-learning framework (as shown in Fig. 3), an agent is connected to its environment via perception and actions. In each step t, the agent observes the system state s_t, chooses an action a_t to perform, and then receives $r(s_t,a_t)$ from the environment and observes new state s_{t+1}. Formally, the model consists of

- a discrete set of environmental states, $S = \{s_t\}$;
- a discrete set of agent actions, $A = \{a_t\}$;
- a reward function $R=\{r(s_t,a_t)\}$: $S \times A \rightarrow R$.

In each state, there is a Q-value associated with each action. The definition of Q-value is the sum of the reinforcements received when performing the associated action and then following the given policy thereafter. Given the definition, it is easy to derive the equivalent of the Bellman equation for Q-learning:

$$Q(s_t,a_t) = r(s_t,a_t) + \gamma \cdot \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) \quad , \tag{1}$$

which is the objective to be maximized in Q-learning. According to this definition, when receiving *reward* in each learning cycle, we update Q-value according to the following equation:

$$Q^{new}(s_t,a_t) = Q(s_t,a_t) + \mu \cdot [r(s_t,a_t) + \gamma \cdot \max_{a_{t+1}} Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)] \quad . \tag{2}$$

Here $r(s_t, a_t)$ is the *reward* received in state s_t with action a_t taken; μ is learning rate; and γ is discount rate. It should be noted that

- the learning rate μ determines what extent the newly acquired information will override the old information to, while the discount rate γ determines the importance of future rewards;
- the number of possible system states and actions must be finite, and as the number of states and actions increases, the Q-table gets bigger and thus the learning accuracy deteriorates quickly;
- if the agent always just takes the action with the highest Q-value for a given state, it might end up in a local maximum, because one action might be repeatedly taken without exploring new actions.

B. State Space

In our Q-learning model, system states are composed of the states of SQ and SP only, because the state of SR is unknown a priori. To simplify the problem, we firstly consider how to describe the state space of a single-core system, and then extend it to multi-core processors.

For single-core processors, we use a vector with two dimensions to describe its state (s_t, q_t). Therein, s_t stands for the processor power state, e.g., run mode or sleep mode. q_t represents the queue status, which indicates how many task requests are stored in queue to wait for processing. Suppose we consider q_t = 0, 1, and 2 respectively for the cases that the number of requests in the queue is 0, 1 and larger than 1. There are as many as $(n_c \cdot n_q)$ states, where n_c is the number of power states, and n_q is the number of queue states. Let (s_{t+1}, q_{t+1}) represent the next state and a_t represent the taken action, we have $(s_t, q_t) \xrightarrow{a_t} (s_{t+1}, q_{t+1})$.

To represent system states in multi-core processors with n cores, we can extend the state vector from two dimensions to $2n$ dimensions. Hence, we have the state representation $(s_{t1}, s_{t2}, ..., s_{tn}; q_{t1}, q_{t2}, ..., q_{tn})$, wherein s_{ti} and q_{ti} are the core power state and the waiting queue state for core i, respectively. With the above representation, however, the size of the state space increases to $(n_c \cdot n_q)^n$. Such a huge state space is a critical problem for learning-based approaches, because in this case many more training samples are needed, and learning accuracy deteriorates quickly. To solve this problem, we utilize neural network to approximate Q-values (detailed later in Section IV).

C. Action Space

In our model, we sample the system state at each time point when a task request arrives. The power manager then observes the current system state, and determines an action for SP to operate. As shown in Fig. 1, when $task_{n+1}$ arrives, its arrival time can determine the time point t_2. At that time point, power manager samples system state, and chooses an action to apply. The action is composed of two components: not only the core that $task_{n+1}$ is assigned to, but also the power state of the assigned core after finishing this task. In other words, the power manager presets the power modes for all the cores. If idle time slots appear in the cores, they will transfer to the appointed power modes.

The action can be represented as ($core_t$,$mode_t$). The variable $mode_t$ stands for the preset mode for assigned core, and $core_t$ is the core index to indicate which core to assign this task to. In this case, the action space size is $(n_c \cdot n)$, where n_c is the number of power modes for each core and n is the number of cores.

D. Reward

The objective of DPM techniques is usually to achieve the maximum power savings at slight performance penalty cost. To achieve a tradeoff between the two items, the *reward* function used in our Q-learning model is expressed as below:

$$R(\vec{s}_t, \vec{a}_t) = -(P(\vec{s}_t, \vec{a}_t) + \beta \cdot RT(\vec{s}_t, \vec{a}_t)) \quad , \tag{3}$$

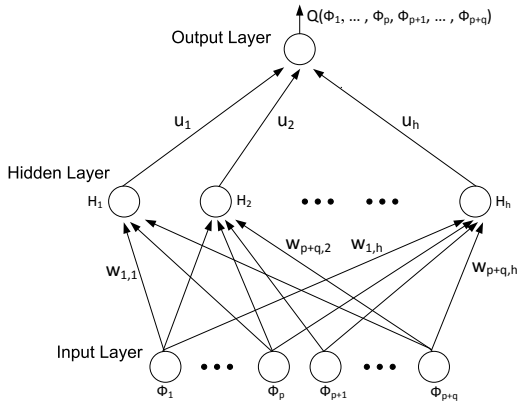

Fig. 4. Multi-Layer Sigmoid Approximation Neural Network

where R is reward, P is mean power dissipation, RT is response time and β is the coefficient to trade off power and performance. If β-value is changed, the weights of mean power and response time in *reward* function are adjusted to satisfy system demand. A larger β-value means that response time is more important to our concern.

At the time point with system state $\vec{s}_t$, the power manager chooses action $\vec{a}_t$. Then the system state transfers from $\vec{s}_t$ to $\vec{s}_{t+1}$, and corresponding reward value can be received.

IV. Proposed Learning Algorithm

A. Q-Function Approximation

One of the most challenging issues in our work is the huge system space size $((n_c \cdot n_q)^n \cdot (n_c \cdot n))$, which is exponentially increased with respect to processor core number n. Q-learning at its simplest version uses tables to store Q-values. This not only costs insufferable memory, but also requires a huge amount of training samples to learn the Q-table accurately. For example, if we describe core with up to 3 power states and 2 queue states, a 8-core processor would have $(3 \cdot 2)^8$ system states and $(3 \cdot 8)$ actions. Suppose that 5 training samples are needed for each table cell, $((3 \cdot 2)^8 \cdot (3 \cdot 8) \cdot 5) = 201,553,920$ training samples are required, which is almost impossible.

To address the above issue, we use neural network (NN) [15] to approximate the Q-function. Hence, the key task in the Q-learning for our problem becomes how to estimate the mapping $Q(\vec{s}, \vec{a}) : \vec{s} \times \vec{a} \rightarrow Q$. We adopt the feedforward neural network to model the mapping $Q(\vec{s}, \vec{a})$ and represent the value function of state-action pair (s_t, a_t). There are a variety of neural networks that are applicable to function approximation, and we consider back propagation neural network (BPNN) [15], one of the most prevailing neural algorithms in dealing with function approximation.

As shown in Fig. 4, there are three layers in the used neural network, namely, input, hidden, and output layers, respectively. In the input layer, the input vector is the composite of state vector ($\vec{s}_1$,...,$\vec{s}_n$) and action vector $\vec{a}$. We use binary encoding scheme to denote the input vectors for every possible system state and action. In the hidden layer, the hidden nodes H_i employs the following sigmoid function,

$$H_j = 1/(1 + e^{-\sum_{i=1}^{p+q} \Phi_i \cdot w_{i,j}}) \quad , \tag{4}$$

where $w_{i,j}$ and u_i are the parameters of neural network, Φ_i denotes one bit of binary input, p denotes the bit number of binary input for system state, while q denotes the bit number of binary input for action. In the output layer, the approximated Q-value function is given by

$$Q(\vec{s}, \vec{a}) = \sum_{i=1}^{h} H_i \cdot u_j \quad . \tag{5}$$

As a whole, this neural network describes a non-linear mapping $Q(\vec{s}, \vec{a})$. At each time t, the parameters of the network ($w_{1,1}$,..., $w_{p+q,h}$; u_1,...,u_h) are updated in a gradient manner with the help of the back-propagation algorithm [15]. The errors propagate backwardly from

the output nodes to the inner nodes to adjust the network's weights. When it is applied to Q-learning, the input of back propagation neural network is the state-action pair and its output is the Q-value corresponding to the state-action pair.

B. Action Selection

The action selection mechanism is an important component of Q-learning. There are two problems to tackle in our action selection phase.

First, if the agent always takes the action with the highest Q-value for a given state, it might end up in a local maximum, because one action might be repeatedly taken without exploring new actions, which prevents us from finding other solution. In other words, action selection may greatly affect learning effectiveness, due to the tradeoff between exploitation and exploration. To balance these two aspects, we employ ε-greedy method for action selection, so that the agent can reinforce the evaluation of the known actions to be good and also explore unknown actions, which helps in avoiding local maximum. We gives the action that owns the highest Q-value a high selected probability (1-ε), and all the actions equally share the remaining probability ε. The probability for choosing a certain action a_i is presented as below.

$$P_i = \begin{cases} (1-\varepsilon)+(\varepsilon/num) & \text{if the Q value of action } a_i \text{ is the highest;} \\ \varepsilon/num & \text{otherwise.} \end{cases} \quad (6)$$

We consider $\varepsilon = 10\%$ here, and *num* is action number. That means, we have the probability of 10% to select another action instead of the action with highest Q-value, to void local maximum.

Second, since one of the motivations in this work is to reduce unnecessary power state transitions to avoid transition costs, our algorithm has the trend to allocate tasks successively to certain cores. This may induce temperature stress on certain cores and cause reliability concerns. To tackle this problem, our action selection mechanism is further modified. Each time, if the temperature of the core chosen is higher than a pre-defined temperature threshold, we would give up this action and try to select another action from the remaining cores by ε-greedy again.

C. Overall Flow

To sum up, the proposed Q-learning algorithm is illuminated in Fig. 5, which starts with initialization (Line 1). The procedure is repeated until there are no more episodes. For every episode, the Q-values are computed via back propagation neural network. We then select an action in the ε-greedy manner (Line 5), and take the action to transfer system state from $\vec{s}$ to $\vec{s'}$ and receive *reward* value (Line 7). When we get the *reward* as feedback, we update the parameters of back propagation neural network using gradient descent algorithm (Line 8), and update state $\vec{s}$ using next state $\vec{s'}$ (Line 9).

Note that, an off-line training phase with a convergence criterion (e.g., the normalized error of approximated Q-value is less than 5%) can be used to improve the solution quality during the beginning of task execution, if necessary. Proposed solution performs online training to both Q-learning and neural network by considering each task as one training sample.

D. Cost Analysis

Our proposed DPM algorithm can be implemented using software and/or extra hardware. For each task, power manager tries to select the proper action before its execution, and receives system information to train the neural network via updating parameters, which induce both energy and performance costs. Through cost analysis, we can estimate that, for each task, a pure-software implementation needs $O(n^2)$ multiplication and addition operations. For example, in 4-core processors with CPU frequency 1*G*Hz, assuming multiplication needs 4 CPU cycles and addition needs 1 CPU cycle, the mean allocation time for each task is in the order of 10μs, which is usually not a big concern as the average task execution time is much higher (e.g., in the order of 10ms). Note that, with an increasing number of processor cores, the cost would increase quadratically, and not be negligible any longer. To tackle this problem, we can easily change our current DPM model to classify cores with similar state into groups, redefine the actions to select a certain group, and then choose a core in the selected group hierarchically.

```
1.  Initialization()
2.      Set neural network parameters to random values
            between 0 and 1.0
3.      Initialize system state s⃗
4.  Repeat for each step of the current episode
5.      Select action a⃗ using ε-greedy
6.      Take action a⃗
7.      Observe next state and receive reward R
8.      Update back propagation neural network parameters
            using gradient descent algorithm
9.      Set system state s⃗ ← s⃗'
10. Until there are no more episodes
```

Fig. 5. Proposed Q-learning Algorithm Based on Back Propagation Neural Network.

```
1.  Repeat for each task arrival
2.      if (there are idle cores, not executing tasks)
3.          Select one of them to execute current task
4.      else
5.          Select the running core with least task number in queue
6.              if (its task number ≤ pre-defined parameter H)
7.                  Use this core to execute current task
8.              else
9.                  Wake up a sleep core to execute current task
10.     Set the selected core into sleep mode when idle period ≥ T_be
11. Until there are no more tasks
```

Fig. 6. Heuristic DPM Approach as Baseline Solution

V. Experimental Results

A. Experimental Setup

To evaluate the effectiveness of proposed DPM solution, we implement a simulator for multi-core processors to obtain the power dissipation and performance index. Hotspot [16] is integrated into our simulator to acquire the operational temperature for processor cores and we set the threshold to avoid allocating task onto a processor core as 85°C. Hypothetical homogeneous 4-core processor and 8-core processor are used in our experiments, wherein all processor cores have the same execution time for a certain task, based on power and thermal features of Intel Atom Processor N450 [17].

As discussed earlier, the proposed DPM policy does not require information about the characteristics of each task, since our concerns in this work are to reduce unnecessary power state transitions (and hence avoid transition costs) and set proper power mode for each processor core. From this viewpoint, the different power value of each individual task does not have a high impact on the effectiveness of the proposed technique. Instead, the task flow in the entire workloads, characterized by task inter-arrival time and task service time, is the decisive factor. Consequently, synthetic workloads are used in our experiments, wherein the power value of each task is generated using PTscalar [18], based on SPEC2000 benchmark programs [19]. The task arrival to the system is assumed as a Poisson process with arrival rate λ, which is widely used in prior work [20], while the service time is an exponential distribution with mean $1/\mu$ in our simulation. By denoting the processor core number as n, task load ρ for each core is $\lambda/n\mu$. A larger ρ-value means that system is running with heavier workloads. Each task set is composed of 1,000 tasks.

There are numerous prior works in the field of dynamic power man-

Parameter ($1/\mu$, ρ)	Target	4-Core Processors						8-Core Processors					
		Oracle	Policy in [12]	Proposed		Δ(%)		Oracle	Policy in [12]	Proposed		Δ(%)	
				β_1	β_2	β_1	β_2			β_1	β_2	β_1	β_2
(0.1, 0.2)	En.	1.170	2.071	1.722	1.724	-16.88	-16.77	1.121	2.061	1.643	1.686	-20.30	-18.19
	RT	0.197	0.283	0.284	0.283	0.41	-0.18	0.189	0.258	0.274	0.259	6.32	0.41
(0.1, 0.4)	En.	1.153	1.990	1.449	1.494	-27.17	-24.91	1.059	1.954	1.311	1.373	-32.88	-29.71
	RT	0.204	0.340	0.300	0.290	-11.80	-14.47	0.187	0.268	0.246	0.242	-8.22	-9.79
(0.1, 0.6)	En.	1.121	1.691	1.333	1.381	-21.14	-18.34	1.066	1.536	1.179	1.265	-23.24	-17.66
	RT	0.201	0.352	0.316	0.292	-10.42	-17.17	0.191	0.311	0.288	0.277	-7.44	-11.04
(0.1, 0.8)	En.	1.060	1.388	1.210	1.224	-12.84	-11.85	1.079	1.385	1.224	1.254	-11.65	-9.45
	RT	0.192	0.356	0.307	0.302	-13.63	-15.17	0.195	0.365	0.293	0.275	-19.79	-24.81

En.: average energy consumption; RT: average response time;

Δ: the difference ratio between the policy in [12] and proposed policy.

TABLE I
ENERGY SAVING/RESPONSE TIME COMPARISON BETWEEN THE POLICY IN [12] AND PROPOSED POLICY

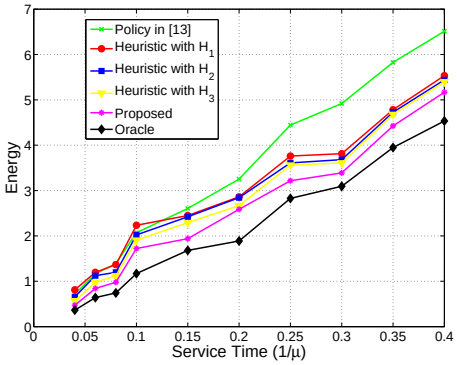

Fig. 7. Energy Consumption with Various Service Times

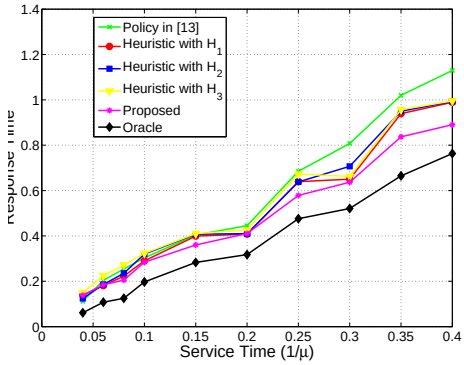

Fig. 8. Response Time with Various Service Times

agement. To the best of our knowledge, however, none of them targets exactly the same problem for multi-core processors as ours. Hence, for comparison, we implement the reinforcement learning-based policy presented in [12], which is the state-of-the-art DPM approach reported in the literature, and used to conduct DPM for each processor core individually. The comparison to this work can demonstrate the advantage of manipulating idle periods globally in multi-core processors, a feature that is not available for single-core processors. In addition, we construct a global heuristic-based DPM approach for multi-core processors as described in Fig. 6 to demonstrate the effectiveness provided by learning. This heuristic-based approach is constructed based on the observation according to our motivational example in Fig. 1. If there are idle cores in the system, assigning tasks onto these cores would not induce any transition costs, hence they should be the first choices; otherwise, we need to choose between cores in run mode and sleep mode. There are not an absolute advantage between these two kinds of cores, hence a pre-defined parameter H is used to decide the choice through observing task queue state. Moreover, We also compare the proposed policy to the *oracle* policy [21], an ideal one that is obtained assuming the task arrivals are known in advance and hence induces no performance penalty.

B. Results and Discussion

In Table I and Table II, Column 1 indicates the core load ρ of the task sets; Column 2 presents the resulting mean energy consumption (*En.*) and mean response time (*RT*) for each task; both 4-core case and 8-core case are included. Each column for presenting proposed policy has two sub-columns for $\beta_1 = 100$ and $\beta_2 = 1,000$, respectively. Here, β_1 and β_2 are the parameter values to trade off power dissipation and response time in Equation 3. The column for heuristic policy in Table II has three sub-columns, showing the results with different parameter values (H = 1, 2 and 3).

First of all, as shown in Table I and Table II, our proposed DPM technique can obtain better energy savings when compared with both learning-based policy [12] and heuristic policy in Fig. 6. To be specific, for the 4-core case with β_2, the proposed policy can achieve 24.91% energy savings with 14.47% response time reduction compared to the single-core learning-based policy in [12], and 16.32% energy savings with 17.19% response time reduction compared with the simple heuristic policy for multi-core processors with $H_2 = 2$, when core load $\rho = 0.4$.

One notable finding from the results is that our DPM technique can provide more power savings when the core load ρ is moderate. This is mainly because when the task sequence is too tight/loose, most of the cores have always been in run/idle mode, thus there are not many power state transitions that can be manipulated by the proposed policy. As for the mean response time, the proposed policy can achieve a considerable decrease in most cases except the case with $\rho = 0.2$. This is expected because in the case of low ρ, the improvement space for response time is limited. If we change heuristic policy parameter H (from $H_1 = 1$ to $H_3 = 3$), heuristic policy can obtain more energy savings at the cost of performance loss. When we increase the tradeoff parameter β (from $\beta_1 = 100$ to $\beta_2 = 1,000$), we can find that energy consumption is increased while the response time is decreased. This is because a larger β-value means that response time is more important in the *reward* function definition (refer to Equation 3). The results of 8-core processors, shown in Table I and Table II, have the similar trend with the 4-core case.

Furthermore, to find out the efficacy of our proposed DPM technique in the cases with various task service times, we keep core load $\rho = 0.2$, and vary task service time from 0.04 to 0.4. As Fig. 7 and Fig. 8 indicate, the proposed policy is the most close to oracle policy and performs much better than the other two kinds of policies.

Finally, we conduct some experiments to obtain the cost of our proposed approach, considering software implementation of the DPM policy. The energy cost is those energy consumed by running our learning algorithm, while the response time cost is the task allocation time. As shown in Table III, the cost of our proposed approach is very

Parameter ($1/\mu$, ρ)	Target	4-Core Processors											8-Core Processors										
		Heuristic			Proposed		$\Delta_1(\%)$		$\Delta_2(\%)$		$\Delta_3(\%)$		Heuristic			Proposed		$\Delta_1(\%)$		$\Delta_2(\%)$		$\Delta_3(\%)$	
		H_1	H_2	H_3	β_1	β_2	β_1	β_2	β_1	β_2	β_1	β_2	H_1	H_2	H_3	β_1	β_2	β_1	β_2	β_1	β_2	β_1	β_2
(0.1,0.2)	En.	2.234	2.021	1.906	1.722	1.724	-22.92	-22.83	-14.82	-14.71	-9.65	-9.55	2.139	1.95	1.775	1.643	1.686	-23.19	-21.18	-15.77	-13.54	-7.44	-5.01
	RT	0.290	0.300	0.313	0.284	0.283	-2.07	-2.41	-5.24	-5.8	-9.27	-9.58	0.27	0.28	0.305	0.274	0.259	1.48	-4.07	-2.09	-7.53	-10.16	-15.08
(0.1,0.4)	En.	1.875	1.785	1.698	1.449	1.494	-22.72	-20.32	-18.84	-16.32	-14.66	-12.01	1.857	1.809	1.8	1.311	1.373	-29.40	-26.06	-27.54	-24.11	-27.17	-23.72
	RT	0.308	0.351	0.375	0.300	0.290	-2.60	-5.84	-14.61	-17.19	-20.00	-22.67	0.241	0.244	0.26	0.246	0.242	2.07	0.41	0.67	-1.05	-5.38	-6.92
(0.1,0.6)	En.	1.698	1.635	1.628	1.333	1.381	-21.50	-18.67	-18.45	-15.56	-18.12	-15.17	1.583	1.502	1.493	1.179	1.265	-25.52	-20.09	-21.51	-15.80	-21.03	-15.27
	RT	0.329	0.345	0.354	0.316	0.292	-3.95	-11.25	-8.58	-15.47	-10.73	-17.51	0.328	0.347	0.345	0.288	0.277	-12.20	-15.55	-17.04	-20.27	-16.52	-19.71
(0.1,0.8)	En.	1.521	1.435	1.430	1.210	1.224	-20.45	-19.53	-15.7	-14.74	-15.38	-14.41	1.548	1.548	1.517	1.224	1.254	-20.93	-18.99	-20.95	-18.98	-19.31	-17.34
	RT	0.365	0.412	0.460	0.307	0.302	-15.89	-17.26	-25.45	-26.79	-33.26	-34.35	0.355	0.381	0.39	0.293	0.275	-17.46	-22.54	-22.98	-27.80	-24.87	-29.49

H_1, H_2, and H_3: heuristic policy with parameter $H = 1, 2$, and 3;

Δ_1, Δ_2, Δ_3: the difference ratio between heuristic policy $H_1/H_2/H_3$ and proposed policy.

TABLE II
ENERGY SAVING/RESPONSE TIME COMPARISON BETWEEN HEURISTIC POLICY IN FIG. 6 AND PROPOSED POLICY

ρ	Allocation Cost in 4-Core Processors					
	Energy	Energy Cost	$\Delta_1(\%)$	Response Time	Time Cost	$\Delta_2(\%)$
0.2	1.722	1.689E-04	0.0098	0.284	3.071E-05	0.0108
0.4	1.449	1.691E-04	0.0117	0.300	3.075E-05	0.0103
0.6	1.333	1.687E-04	0.0127	0.316	3.068E-05	0.0097
0.8	1.210	1.688E-04	0.0140	0.307	3.069E-05	0.0100
ρ	Allocation Cost in 8-Core Processors					
	Energy	Energy Cost	$\Delta_1(\%)$	Response Time	Time Cost	$\Delta_2(\%)$
0.2	1.643	5.094E-04	0.0310	0.274	9.262E-05	0.0338
0.4	1.311	5.101E-04	0.0389	0.246	9.274E-05	0.0377
0.6	1.179	5.091E-04	0.0432	0.288	9.257E-05	0.0321
0.8	1.224	5.093E-04	0.0416	0.293	9.259E-05	0.0316

ρ, task load for one core; Δ_1, energy cost ratio; Δ_2, time cost ratio.

TABLE III
ALLOCATION COST IN 4-CORE & 8-CORE PROCESSORS

small and can almost be negligible. In 4-core processors, the mean energy cost is less than 0.02% of mean energy consumption, while the response time cost is about 0.01% of mean response time. In 8-core processors, the cost is also quite small, about 0.04% in both energy and response time.

VI. CONCLUSION AND FUTURE WORK

Power consumption is a key issue in the design of computing systems today, especially in portable devices which are more sensitive to battery life. In this paper, by including task allocation into our learning-based DPM framework for multi-core processors, we are able to manipulate idle periods on processor cores to achieve a better trade-off between power consumption and system performance. Experimental results based on synthetic workloads prove the effectiveness of our proposed approach significantly.

Our proposed approach can work quite well in 4-core and 8-core processors. However, since the cost of our proposed approach is increased quadratically with respect to the number of processor cores, if the number of cores becomes huge, the allocation cost may not be negligible. For instance, it can be estimated that the allocation cost is about 1% for 64-core processors, and about 3% for 100-core processors. To tackle this problem, we need to work out new definitions of system states and actions to improve the algorithm scalability in the future. One possible method is to group cores with the same core state together and change the action from core selection to group selection. This approach will be further explored in our future research work.

In addition, for the sake of simplicity, we focus on dynamic power consumption and assume no dependencies among tasks in this work, even though our proposed learning-based framework is applicable after lifting these assumptions. We plan to investigate the impact of leakage power and task dependencies on our proposed framework in the future as well.

VII. ACKNOWLEDGEMENT

This work was supported in part by the General Research Fund CUHK418708 and CUHK418111 from Hong Kong SAR Research Grants Council (RGC), and in part by a grant N_CUHK417/08 from the NSFC/RGC Joint Research Scheme.

VIII. REFERENCES

[1] L. Benini, A. Bogliolo, G. Paleologo, and G. De Micheli. Policy Optimization for Dynamic Power Management. *IEEE Transactions on Computer-Aided Design*, 18(6):813–833, June 2001.

[2] C. Hwang and A. Wu. A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation. In *Proc. Intl. Conf. on Computer-Aided Design*, pp. 28–32, 1997.

[3] T. Simunic, L. Benini, and G. Micheli. Event-Driven Power Management of Portable Systems. In *Intl. Symposium on System Synthesis*, pp. 18–23, 1999.

[4] G. Dhiman and T. Rosing. Dynamic Power Management Using Machine Learning. In *Proc. Intl. Conf. on Computer-Aided Design*, pp. 747–754, 2006.

[5] H. Jung and M. Pedram. Improving the Efficiency of Power Management Techniques by Using Bayesian Classification. In *Proc. Intl. Symposium on Quality of Electronic Design*, pp. 178–183, 2008.

[6] C. Steinbach. A Reinforcement Learning Approach to Power Management. *M.Eng Thesis, Massachusetts Institute of Technology*, May 2002.

[7] A. Karlin, M. Manesse, L. McGeoch, and S.Owicki. Competitive Randomized Algorithms for Nonuniform Problems. In *Algorithmica*, pp. 542–571, 1994.

[8] M. Srivastava, A. Chandrakasan, and R. Brodersen. Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Compuation. *IEEE Transactions on VLSI Systems*, 4(1):42–55, March 1996.

[9] G. Theocharous, et al. Machine Learning for Adaptive Power Management. *Intel Technology Journal*, 2006.

[10] C. Isci, et al. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Proc. of Intl. Symposium on Microarchitecture*, pp. 347-358, 2006.

[11] H. Jung and M. Pedram. Supervised Learning Based Power Management for Multicore Processors. *IEEE Transactions on Computer-Aided Design*, 29(9):1395 - 1408, September 2010.

[12] Y. Tan, W. Liu, and Q. Qiu. Adaptive Power Management Using Reinforcement Learning. In *Proc. Intl. Conf. on Computer-Aided Design*, pp. 461–467, 2009.

[13] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proc. Intl. Symposium on Low Power Electronics and Design*, pp. 38-43, 2007.

[14] M. Ghasemazar, E. Pakbaznia, and M. Pedram. Minimizing the power consumption of a Chip Multiprocessor under an average throughput constraint. In *Proc. of Intl. Symposium on Quality Electronic Design*, pp. 362-371, 2010.

[15] Stephen Marsland. Machine learning: an algorithmic perspective. *Boca Raton : CRC Press*, c2009.

[16] K. Skadron, et al. Temperature-Aware Microarchitecture. In *Proc. of the 30th Intl. Symposium on Computer Architecture*, 2003.

[17] Intel Atom Processor N400 & N500 Series Datasheet, http://www.intel.com/products/processor/atom

[18] W. Liao, L. He, and K.M. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. In *IEEE Transactions on Computer-Aided Design*, pp. 1042-1053, 2005.

[19] SPEC2000, http://www.spec.org

[20] B. S. Yoo and C. R. Das. A Fast and Efficient Processor Allocation Scheme for Mesh-connected Multicomputers. *IEEE Transactions on Computers*, 51(1):46–60, January 2002.

[21] E.Y. Chung, L. Benini, A. Bogliolo, Y.H. Lu, and G.D. Micheli. Dynamic power management for nonstationary service requests. In *IEEE Transactions on Computers*, pp. 1345-1361, 2002.