# Scalable Power Management Using Multilevel Reinforcement Learning for Multiprocessors

GUNG-YU PAN, National Chiao Tung University
JING-YANG JOU, National Central University and National Chiao Tung University
BO-CHENG LAI, National Chiao Tung University

Dynamic power management has become an imperative design factor to attain the energy efficiency in modern systems. Among various power management schemes, learning-based policies that are adaptive to different environments and applications have demonstrated superior performance to other approaches. However, they suffer the scalability problem for multiprocessors due to the increasing number of cores in a system. In this article, we propose a scalable and effective online policy called MultiLevel Reinforcement Learning (MLRL). By exploiting the hierarchical paradigm, the time complexity of MLRL is $O(n \lg n)$ for $n$ cores and the convergence rate is greatly raised by compressing redundant searching space. Some advanced techniques, such as the function approximation and the action selection scheme, are included to enhance the generality and stability of the proposed policy. By simulating on the SPLASH-2 benchmarks, MLRL runs 53% faster and outperforms the state-of-the-art work with 13.6% energy saving and 2.7% latency penalty on average. The generality and the scalability of MLRL are also validated through extensive simulations.

Categories and Subject Descriptors: D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*; I.2.6 [**Artificial Intelligence**]: Learning—*Parameter learning*; J.6 [**Computer-Aided Engineering**]: Computer-Aided Design (CAD)

General Terms: Design, Algorithms, Performance, Management

Additional Key Words and Phrases: Dynamic power management, multiprocessors, reinforcement learning

**ACM Reference Format:**
Gung-Yu Pan, Jing-Yang Jou, and Bo-Cheng Lai. 2014. Scalable power management using multilevel reinforcement learning for multiprocessors. ACM Trans. Des. Autom. Electron. Syst. 19, 4, Article 33 (August 2014), 23 pages.
DOI: http://dx.doi.org/10.1145/2629486

## 1. INTRODUCTION

Power consumption has become the bottleneck for digital designs in the past decade [Pedram 1996]. Among many low-power techniques, online power saving is widely applied to appliances that pose stringent energy requirements, such as battery-powered embedded devices [Jha 2001]. Dynamic power management (DPM) is one of the most popular power saving approaches. When a device is idle, it can be switched into inactive states to avoid wasting power [Benini et al. 2000]. The power states are defined in the Advanced Configuration and Power Interface (ACPI) [2011] and supported by many modern commercial systems, such as Enhanced Intel SpeedStep Technology (EIST) [Intel 2013], AMD PowerNow! [AMD 2013], ARM Intelligent Energy Controller (IEC)

This work is supported by the National Science Council under grant NSC101-2221-E-008-137-MY3.
Authors' addresses: G.-Y. Pan (corresponding author), Department of Electronics Engineering, National Chiao Tung University, 1001 University Road, Hsinchu City, Taiwan; email: astro@eda.ee.nctu.edu.tw; J.-Y. Jou, Department of Electrical Engineering, National Central University, Taoyuan County, Taiwan; B.-C. Lai, Institute of Electronics, National Chiao Tung University, 1001 University Road, Hsinchu City, Taiwan.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2014 ACM 1084-4309/2014/08-ART33 $15.00
DOI: http://dx.doi.org/10.1145/2629486

[ARM 2005], and MIPS Cluster Power Controller (CPC) [Knoth 2009]. With the platforms supporting the interfaces, software-based power managers are able to determine the power states for the devices.

Recently, multiprocessor systems have become the mainstream in both general-purpose and embedded computers. The number of processors in a system grows as time advances [Held et al. 2006]; it is expected that there will be hundreds of processors in a system in the near future. Since traditional DPM policies are mostly proposed for uniprocessor systems, it is imperative to design an effective policy suitable for multiprocessor systems.

### 1.1. Dynamic Power Management for Multiprocessor Systems

For a multiprocessor system, some of the cores may not be fully utilized but idle or with low utilization rates, hence applying power management is crucial to avoid wasting energy. Different from traditional approaches, the *efficiency* and the *effectiveness* of a power manager need to be extraordinarily considered for a multiprocessor system. For complex policies, the execution overheads become serious when there are more cores in a system. Even worse, the policies become less effective as the solution space grows larger. Thus, we need to design a low-complexity policy to ensure the scalability and enhance the solution quality for multiprocessor systems.

The reinforcement learning technique is applied to multiprocessors in Ye and Xu [2012] and shown superior than the distributed power managers using Tan et al. [2009]. All the core states and actions are encoded and the learning function is approximated using the back-propagation neural network (BPNN). Tasks are assumed independent, thus their policy is able to not only determine the power mode, but also assign each task to a specific core. However, task assignment (context scheduling) has other considerations, such as data locality, that greatly affect the performance in real environments. Besides, scalability is still a problem for its time complexity $O(n^2)$.

### 1.2. Our Contributions

In this article, we propose a policy called MultiLevel Reinforcement Learning (MLRL) that is much more scalable in efficiency and effectiveness for multiprocessors. The hierarchical approach performs better than both of the distributed policies lacking the global views and the centralized policies suffered the exponentially increasing solution space. Besides, we choose another low-overhead network for function approximation and the adaptive approach for action selection. The proposed policy is general and applicable in multiprogrammed or multithreaded environments without preknowledge to train the policy beforehand. It is evaluated using real benchmarks on the cycle-accurate simulator.

The main novelty of this article is the carefully designed multilevel framework that resolves the scalability issue by turning the exponential decision problem into linear, and provides the knobs for further optimizations. Both the solution quality and the policy efficiency are greatly raised. Moreover, the problem formulation is more general and the effectiveness is evaluated through real benchmarks. In short, we make the following contributions.

—The multilevel paradigm is exploited to compress the searching space, speed-up the convergence rate, and result in $O(n \lg n)$ time complexity for $n$ cores.
—The proposed online policy is independent of context scheduling; it neither needs preliminary information of the workload nor assumes the tasks are independent.
—The simulation results show that our policy runs 53% faster and outperforms the state-of-the-art work with 13.6% energy saving and 2.7% latency penalty on average

for the SPLASH-2 benchmarks; the performance penalty is close to zero while the energy saving is close to the upper bound (14.7%).

The remainder of this article is organized as follows. Section 2 explains the background approaches of our policy and Section 3 formulates the problem. Then Section 4 illustrates the basic framework of our policy and Section 5 provides several enhancement techniques. Afterward, Section 6 shows the simulation results and examines both the basic and advanced approaches. Lastly, Section 7 surveys the related works and Section 8 summarizes this article.

## 2. BACKGROUND

In this section, the background of the two key concepts is introduced before describing our policy. The same notations are inherited in this article.

### 2.1. Reinforcement Learning

Reinforcement learning (RL) [Barto and Mahadevan 2003] is applied in some previous works [Tan et al. 2009; Ye and Xu 2012], outperforming other DPM approaches. There are some strong similarities between DPM and RL: the *manager* of DPM decides the next power state according to the current system status and the past statistics, while the *agent* of RL observes the environment state $s_t$ at time $t$ (called an *epoch*), takes an action $a_t$, and accordingly receives the reward $r_t$. Because the trial-and-error processes are similar, the power manager can be implemented using an RL-based agent.

One of the most effective RL algorithms is Q-learning, which keeps a Q-value for every state-action pair. The Q-value $Q(s_t, a_t)$ is the average reward prediction of the pair $(s_t, a_t)$. During the *decide* phase, the agent selects the action $a_t$ based on the Q-values of the current state $s_t$ and then updates $Q(s_t, a_t)$ after receiving the reward $r_t$ in the *update* phase. Note that the agent may not be able to receive $r_t$ right at time $t$ but in the future.

There are three basic methods to decide $a_t$: greedy, $\epsilon$-greedy, and softmax. The greedy method picks the action with the highest Q-value. The $\epsilon$-greedy strategy also selects the action with the highest Q-value most of the time, but may select other actions with a prespecified small probability $\epsilon$. The softmax method assigns the probability of selecting each action proportional to $\exp(Q(s_t, a_t)/\tau)$, where $\tau$ is the temperature that decreases as time advances.

The Q-value is updated according to the equation

$$Q(s_t, a_t) \xleftarrow{\text{update}} Q(s_t, a_t) + \mu\left(r_t + \gamma \cdot \max_{a'} Q(s', a') - Q(s_t, a_t)\right), \tag{1}$$

where $0 \le \mu \le 1$ is the learning rate and $0 \le \gamma < 1$ is the discount rate that accounts for future rewards, and $s'$ and $a'$ are the next state and action after taking $a_t$, respectively.

Since an RL agent learns online, the *convergence time* (number of epochs for the Q-values becoming stable) should be minimized, otherwise the agent would make inferior decisions. Besides, the agent should be efficient to minimize the *runtime overhead* (the time to execute the agent). When the state space (all the possible states) is large, more samples are needed for training before convergence. When the action space (all the possible actions with respect to the specific state) is large, the determination overhead is proportionally increased. Therefore, policy designers need to avoid the explosion of the state and the action spaces.

### 2.2. MultiLevel Paradigm

The multilevel paradigm is first proposed in Karypis et al. [1999] for circuit partitioning and then applied to other VLSI problems. During the *coarsening* phase, the elements are recursively clustered until the problem size is small enough to be solved. Then in the *uncoarsening* phase, the coarsened elements are declustered while the refinement algorithm is applied. A coarse solution is produced between the two phases.

Algorithms based on the multilevel paradigm are able to produce high-quality solutions in a small amount of time, while flat algorithms face lots of small elements thus lack of global views over the problems. The coarsening phase generates a good approximation of the original problems, so better initial solutions can be obtained. Then in the uncoarsening phase, the refinement algorithms are more effective because they are able to focus on local problems with smaller sizes.

## 3. PROBLEM FORMULATION

### 3.1. System Model

The target architecture is a multiprocessor system containing $n$ homogeneous cores and $m$ threads per core. It is able to simultaneously execute $n \times m$ contexts. Since our focus is on the scalability issues of multiprocessors instead of other peripherals or subsystems, only the multiprocessor cores are considered in the rest of this article.

The power manager is implemented in the operating system and activated periodically with period $T$ [Isci et al. 2006; Winter et al. 2010]. It is able to switch the cores into different power modes that are given statically according to ACPI [2011]. The mode switching is done in per-core granularity due to higher power saving ratios [Sharkey et al. 2007; Kim et al. 2008]; because the power modes cannot be switched instantaneously, some transition overheads including extra delay and energy are induced. Physical information is available for the power manager, measured by performance counters and power sensors [Isci et al. 2006; Ye and Xu 2012].

Since the environments and applications are diverse for different multiprocessor systems, the power manager should be designed for general applications. The workload characteristics are assumed unknown in advance; it may comprise several single- or multithreaded programs. To tackle various workloads, the context schedulers have other complex considerations, such as data locality and load balancing. Hence, the power manager is assumed to be general and independent of the scheduling policies.

### 3.2. Problem Statement

In general, DPM policies have three optimization goals [Benini et al. 2000]. First, the total energy saving should be maximized. Note that lowering power consumption does not imply energy saving in some circumstances, because the latency may be longer. Second, the latency penalty is minimized, even when the energy is lowered. Third, the runtime of the manager should be minimized to avoid prolonging the kernel time.

The focus of this article is designing the DPM policy for multiprocessor systems according to the preceding system model and optimization goals. Since the proposed policy is based on reinforcement learning, the terminologies are described as follows.

### 3.3. Learning-Based Power Management

The overall flow of learning-based power management is shown in Figure 1 with the timeline in Figure 2. At each epoch $t$, according to the current state $s_t$, the agent (power manager) decides the action $a_t$ based on the Q-values. In addition, $Q(s_{t-1}, a_{t-1})$ is updated using (1) with past reward $r_{t-1}$ in $[t-1, t]$ (the current reward $r_t$ cannot be received immediately until $t+1$). When the agent is activated at $t = 1$, $Q(s_0, a_0)$ is updated using $r_0$ and then the action $a_1$ is taken.

To apply the Q-learning algorithm for dynamic power management, the current state is a pair $s = (sp, sq)$ with $sp$ for the current power mode and $sq$ for the number of tasks

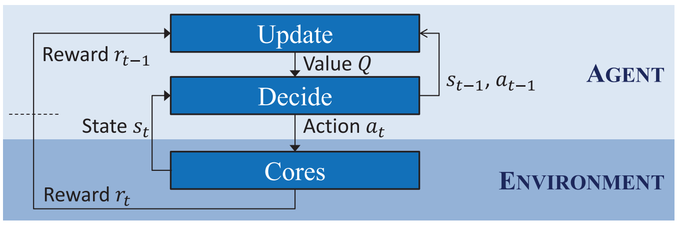

Fig. 1. Overall flow of power management based on Q-learning.

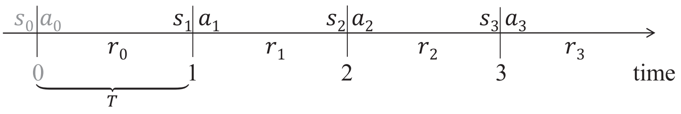

Fig. 2. Timeline (epochs) of power management based on Q-learning.

Table I. List of Notations

| Notation | Meaning |
|---|---|
| $n$ | Number of total processors |
| $m$ | Maximum number of tasks per processor |
| $T$ | Power management period |
| $s_t$ | System state at time $t$ |
| $sp_t$ | Power state (computation capability) at time $t$ |
| $sq_t$ | Queue state (number of tasks in queues) at time $t$ |
| $a_t$ | Target power state (computation capability) at time $t$ |
| $r_t$ | Lagrangian figure-of-merit for the state-action pair $(s_t, a_t)$ |
| $\beta$ | Trade-off parameter between performance and power |
| $\mu$ | Learning rate of reinforcement learning |
| $\gamma$ | Discount rate of reinforcement learning |
| $\epsilon$ | The probability threshold of escaping from the greedy choice |
| $\phi_k$ | The $k$th hidden node in a Radial Basis Function (RBF) network |
| $\boldsymbol{u_k}$ | The position vector of $\phi_k$ in an RBF network |
| $\sigma_k$ | The width of $\phi_k$ in an RBF network |
| $\alpha_k$ | The weight of $\phi_k$ in an RBF network |
| $K$ | The number of hidden nodes in an RBF network |
| $e_{min}$ | Desired accuracy of an RBF network |
| $\delta$ | Distance between nodes in an RBF network |
| $\lambda$ | The decay constant of $\delta$ in an RBF network |
| $\kappa$ | The overlap factor between hidden nodes in an RBF network |

in queue(s), and the action $a$ is the target power mode. The reward function

$$r = \beta \cdot throughput - (1 - \beta) \cdot power \tag{2}$$

is the Lagrangian figure-of-merit for instantaneous throughput and power where $0 \leq \beta \leq 1$ is the trade-off parameter. Although optimizing instantaneous throughput does not guarantee optimum overall latency, this greedy strategy is a must when the incoming workloads are unknown [Mariani et al. 2011].

The important notations are listed in Table I. As stated before, the first and second sectors list the notations and parameters for the learning-based power management described in Section 4, respectively. The third sector lists the parameters for some enhancement techniques defined in Section 5. The values of the parameters are set for simulations in Section 6, obtained from previous works.

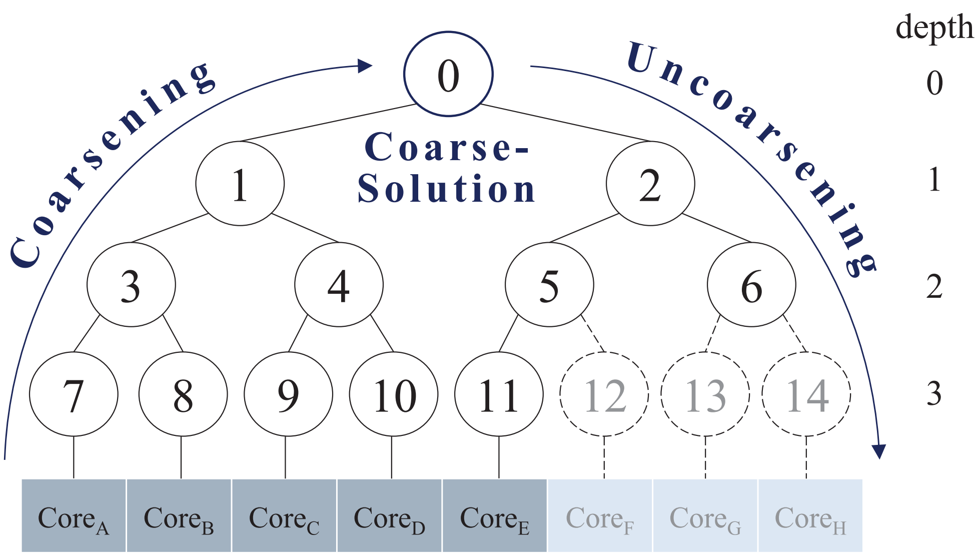

Fig. 3. A binary tree for the multilevel framework on top of a multiprocessor.

## 4. MULTILEVEL REINFORCEMENT LEARNING

In this section, the basic power management framework is described, while some advanced techniques are provided in the next section to further enhance the performance of the proposed policy.

### 4.1. The MultiLevel Framework

Since there are at least $2^n$ states and actions for per-core DPM, directly applying Q-learning to multiprocessors is not scalable due to the explosion of policy runtime. Moreover, the agent may find inferior solutions because of the huge searching space [Barto and Mahadevan 2003]. Therefore, our learning algorithm is based on the multilevel paradigm to reduce the overhead and shrink the searching space to enhance the solution quality.

In order to implement the coarsening and uncoarsening phases in the multilevel paradigm, a binary tree is built on top of the cores, as shown in Figure 3 where the number of cores is denoted as $n$ and $D = \lceil \lg n \rceil$ is the depth of the tree. The vertexes are numbered from the root to the rightmost leaf. For a general multiprocessor system, the number of cores $n$ could be either power-of-two or not; a *complete* binary tree is built on top of it for general cases (five cores and eleven vertexes in Figure 3), while the tree becomes *full* for power-of-two cores (eight cores and fourteen vertexes in Figure 3). In general, there are three cases for a vertex: containing two children, containing only the left child, or containing no child.

In the proposed MultiLevel Reinforcement Learning (MLRL) framework, a vertex $v$ contains six attributes $(sp_{t-1}, sp_t, sq_{t-1}, sq_t, a_t, r_{t-1})$: $sp_{t-1}[v]$, $sp_t[v]$, $a_t[v]$ represent the past, current, and target aggregate power state, respectively, $sq_{t-1}[v]$ and $sq_t[v]$ represent the past and current queue state, respectively, and $r_{t-1}[v]$ represents the received Lagrangian reward, calculated according to (2) with respect to the past state-action pair. A parent vertex represents the coarse version of its children, thus the root represents the coarse version of the entire multiprocessor system. For a homogeneous system, the attributes of a parent vertex are the summation of the attributes of its children.

**ALGORITHM 1:** MultiLevel-Reinforcement-Learning

```
1  for $d \leftarrow D$ to 1 do
2      foreach vertex $v$ with $depth(v) = d$ do
3          Update-Node($v$);
4      end
5  end
6  Update-Root();
7  Decide-Root();
8  for $d \leftarrow 0$ to $D - 1$ do
9      foreach vertex $v$ with $depth(v) = d$ do
10         Decide-Node($v$);
11     end
12 end
```

The overall flow of the proposed policy is described in Algorithm 1. There are three steps in our algorithm: the coarsening phase (lines 1–5), the coarse solution phase (lines 6–7), and the uncoarsening phase (lines 8–12). First, the attributes are collected from the leaves to the root and the Q-values are updated in `Update-Node`. Then the coarse solution is made on the root in `Update-Root` and `Decide-Root`; the basic approaches are first described for the coarse solution phase in this section, while some enhancement techniques are introduced in the next section. At last, the fine-grained solutions are decided hierarchically in `Decide-Node`, according to the coarse-grained solution of each node and the Q-values of its children.

Overall, first the total number of active cores and tasks (ready or running) are obtained, respectively, and the Q-values are updated simultaneously. Then, the number of active cores of the next period is accordingly determined. At last, these resources are distributed to the individual cores. The details of the three phases are explained in the following three sections, respectively. At last, the time complexity is analyzed in the last section.

## 4.2. The Coarsening Phase

The coarsening phase starts from the leaves (with depth $D$) to the branches under the root (with depth 1). There are two missions in this phase: collecting the attributes $(sp_{t-1}, sp_t, sq_{t-1}, sq_t, r_{t-1})$ and updating the Q-values. Note that the attribute value of the action $a_t$ is determined in the uncoarsening phase.

For each leaf node $v$, the current power and queue states are sampled from the corresponding core as $sp_t[v]$ and $sq_t[v]$, respectively, the past states $sp_{t-1}[v]$ and $sq_{t-1}[v]$ are stored in the last epoch, and the reward $r_{t-1}[v]$ is calculated according to (2) using the received power and performance values.

Otherwise, for each branch node $v$, the attributes $(sp_{t-1}, sp_t, sq_{t-1}, sq_t, r_{t-1})$ are summed up from its left child $vl$ and right child $vr$, so that the attributes represent the aggregate behavior of the descent leaf nodes. For example, $sp_t[v]$ represents the current number of active processors in the subtree of $v$. Note that this coarsening methodology greatly reduces the data size from exponential to linear while keeping the principal information. The coarsened granularity is resolved in the uncoarsening phase.

Then the Q-values are updated using the attributes $(sp_{t-1}, sp_t, sq_{t-1}, sq_t, r_{t-1})$. For each vertex $v$ on depth $d$, one of its Q-values (denoted as $Q^{(v)}$) is updated according to (1) for the state-action pair $(s_{t-1}, a_{t-1})$. The state $s_{t-1} = (sp_{t-1}, sq_{t-1})$ and the action $a_{t-1} = sp_t$ are stored in the last epoch. The maximum-possible future Q-value

$$maxQ \leftarrow \max_{0 \le a' \le n/2^d} Q^{(v)}(sp_t[v], sq_t[v], a') \tag{3}$$

is calculated using the sampled current state $(sp_t[v], sq_t[v])$ instead of estimating $s'$ in (1). Then the Q-value is updated as

$$Q^{(v)}(sp_{t-1}[v], sq_{t-1}[v], sp_t[v]) \xleftarrow{\text{update}} Q^{(v)}(sp_{t-1}[v], sq_{t-1}[v], sp_t[v]) \\ + \mu \cdot \big(r_{t-1}[v] + \gamma \cdot maxQ - Q^{(v)}(sp_{t-1}[v], sq_{t-1}[v], sp_t[v])\big). \quad (4)$$

In the general non-power-of-two architectures, we need to ensure that the collected attributes $(sp_{t-1}, sp_t, sq_{t-1}, sq_t, r_{t-1})$ of any node represent the aggregate behavior of its descent subtree. For a parent vertex $v$ with only the left child $vl = 2v + 1$ (such as node 5 in Figure 3 for the five-core system), its attributes are copied from the child. For a parent vertex with no child (such as node 6 in Figure 3 for the five-core system), its attributes are all zero. For a parent vertex with two children, the updating process is the same as that of the power-of-two architectures. Using these rules, any vertex $v$ still represents the aggregate behavior of its descent subtree.

### 4.3. The Coarse Solution Phase

There are two main steps in this phase. `Update-Root` is similar to the coarsening phase where the attributes of the overall system are accumulated and one of the Q-values $Q^{(0)}$ is updated. Then the target number of active processors is determined by the action selection scheme in `Decide-Root`, according to the Q-values $Q^{(0)}$.

The `Update-Root` function first obtains the attributes $(sp_{t-1}, sp_t, sq_{t-1}, sq_t, r_{t-1})$ on the root node. These attributes represent the coarsened behavior of the overall system; they provide the global view (aggregate active resources, loading, and reward) but lack details (the distribution of these attributes on the cores). Note that, besides the contexts allocated to cores, the number of unallocated (ready) contexts are added into $sq_t[0]$ as well to represent the overall system loading.

Since the candidates of actions on the root can be an arbitrary number of active processors $0 \leq a' \leq n$ regardless of the current power state $sp$, the state $s = (sp, sq)$ is approximated using the number of tasks $sq$ to focus on steady-state rewards. In other words, the amount of computation resources is determined according to the system loading in this phase, while the transition costs are optimized in the uncoarsening phase. In fact, the accumulated transition costs are the same, regardless the switching orders. For example, if allocating three active cores is the best solution for current system loading while there is only one active core currently, the accumulated transition costs are the same between turning on two more cores directly and turning on one more core in two epochs.

Thus, the Q-values on the root contain only two dimensions. The updating process is reduced to

$$Q^{(0)}(sq_{t-1}[0], sp_t[0]) \xleftarrow{\text{update}} Q^{(0)}(sq_{t-1}[0], sp_t[0]) \\ + \mu \cdot \left(r_{t-1}[0] + \gamma \cdot \left(\max_{0 \leq a' \leq n} Q^{(0)}(sq_t[0], a')\right) - Q^{(0)}(sq_{t-1}[0], sp_t[0])\right). \quad (5)$$

Since the state space is shrunk from $(n+1) \times (mn+1)$ to $(mn+1)$ while the action space remains $(n+1)$, the convergence time is greatly reduced. The deciding process is also based on the values of $Q(sq, a')$.

The $\epsilon$-greedy method is taken as the basic action selection scheme

$$a_t[0] \leftarrow \begin{cases} \arg\max_{0 \leq a' \leq n} \left(Q^{(0)}(sq_t[0], a')\right) & \text{if } \xi > \epsilon \\ \text{random number from } [0, n] & \text{otherwise,} \end{cases} \quad (6)$$

where $0 \leq \epsilon \leq 1$ is the small probability to escape from the local optimum and $0 \leq \xi \leq 1$ is the uniform random number that changes every time. Because the initial Q-values are the same, the smallest value of $a_t[0]$ (with the same Q-values) is selected to break ties. Although the softmax method is more delicate in action selection than $\epsilon$-greedy, it is difficult to preset a proper value to the parameter $\tau$ without knowledge of the application [Sutton and Barto 1998].

The system behavior is strongly related to the value of $\epsilon$. When $\epsilon$ is small, the system is stable but often trapped in the local optimum. In the extreme case, the system may be stuck on the initial full-on state ($a_t[0] = n$) if the pure greedy ($\epsilon = 0$) strategy is taken. On the other hand, the system may usually jump to random inferior states when $\epsilon$ is large. This is known as the dilemma of exploration and exploitation [Tokic and Palm 2011]. Note that the $\epsilon$-greedy action selection scheme is only our basic strategy where it is used in Ye and Xu [2012] as well, while some advanced action selection schemes are explained in the next section.

### 4.4. The Uncoarsening Phase

The uncoarsening phase starts from the root (with depth 0) to the deepest branches (with depth $D - 1$). The mission is to hierarchically distribute the aggregate resources (target number of active cores $a_t[0]$) to the distinct cores and restore the granularity. The decisions are made according to the attributes and Q-values updated in the coarsening phase.

For each node $v$ with depth $d$, the number of active cores $a_t[v]$ is distributed to $a_t[vl]$ and $a_t[vr]$, where $vl$ and $vr$ are the left and right child nodes, respectively, such that $a_t[vl] + a_t[vr] = a_t[v]$. Let $l[v]$ denote the number of leaves in the descent subtree of each vertex $v$ (where $l[0] = n$), thus the target number of active cores is limited by $a_t[v] \leq l[v]$. According to their current power mode $sp_t[vl], sp_t[vr]$ and number of running tasks $sq_t[vl], sq_t[vr]$, all of the combinations are tried and evaluated by the summation of their Q-values as

$$(a_t[vl], a_t[vr]) \leftarrow \arg\max_{0 \leq al' \leq l[vl], 0 \leq ar' \leq l[vr], al'+ar'=a_t[v]} \big(Q^{(vl)}(sp_t[vl], sq_t[vl], al') + Q^{(vr)}(sp_t[vr], sq_t[vr], ar')\big). \quad (7)$$

The pair with the largest summed Q-value is chosen.

Note that the transition costs are considered implicitly in (7). The transition delay of switching the power modes lowers the number of committed instructions, which is revealed in the rewards and Q-values. For example, it is expected that $Q^{(vl)}(1, 1, 1) + Q^{(vr)}(0, 0, 0) \geq Q^{(vl)}(1, 1, 0) + Q^{(vr)}(0, 0, 1)$ after convergence, because of the transition power of turning off the cores and the transition delay of turning on the cores. Additional performance overheads, such as context (data) migration and refilling of cache and pipeline, are counted in the rewards and Q-values as well.

For systems supporting multiple power-down modes, the distributed learning policy [Tan et al. 2009] is applied at the end of this phase. For each core, if it is turned off in the uncoarsening phase, then one of the off states is chosen according to the static power and transition cost.

### 4.5. Time Complexity Analysis

In the coarsening phase, the max operation in (3) of `Update-Node` requires $n/2^d + 1$ comparisons for each node on depth $d$. The coarsening process goes from $d = \lg(n)$ to $d = 1$ with $2^d$ nodes in depth $d$, so the time complexity of the coarsening phase is $\sum_{d=1}^{\lg(n)} (2^d(n/2^d + 1)) = \sum_{d=1}^{\lg(n)} (n + 2^d) = O(n \lg n)$.

In the coarse solution phase, both of the max operations in (5) of `Update-Root` and the arg max operation (6) of `Decide-Root` require $n + 1$ comparisons. Therefore, the time complexity of the coarse solution phase is $O(n)$.

In the uncoarsening phase, the arg max operation in (7) of `Decide-Node` requires $n/2^{d+1} + 1$ comparisons and there are $2^d$ nodes in depth $d$, so the time complexity of the uncoarsening phase starting from $d = 0$ to $d = \lg(n) - 1$ is $\sum_{d=0}^{\lg(n)-1} 2^d(n/2^{d+1} + 1) = O(n \lg n)$.

According to the previous analysis, the overall time complexity is $O(n \lg n)$ for multiprocessors with power-of-two cores. In general cases, the time complexity is bounded by $O(n' \lg n')$, where $n' = 2^{\lceil \lg n \rceil}$. Because $n \leq n' < 2n$, the time complexity of general architectures is still $O(n \lg n)$.

## 5. ENHANCEMENT TECHNIQUES

In the previous section, the multilevel paradigm is applied to traditional reinforcement learning so that the searching space is greatly reduced while preserving the global view. There are some characteristics of power management on multiprocessors that can be exploited to further enhance the proposed policy.

First, the agent is able to estimate the reward of an state-action pair without visiting it several times because the relation between active resources and instantaneous power (and performance) is continuous. Thus, the coarse solution is estimated on the root using modified Q-learning with function approximation (interpolation) in `Update-Root`.

Second, the Q-values converge when the system is stable while they change rapidly at the beginning or when the system is unstable. Using the smart action selection scheme to detect the system stability in `Decide-Root`, the random action is selected when the system is unstable while the greedy choice is taken after convergence.

### 5.1. Function Approximation with the Radial Basis Function Network

There are $(mn + 1) \times (n + 1)$ state-action pairs in total for Q-learning, and each pair requires several training samples. This becomes a scalability problem when the system has lots of cores (large $n$) and deep multithreading (large $m$). Large searching space implies slow convergence and poor generality in learning, and leads to inferior results. Therefore, function approximation (by supervised learning) is widely used in reinforcement learning schemes [Tham 1994].

The radial basis function (RBF) is chosen to approximate our reinforcement learning scheme due to faster convergence rate with simpler structure [Wu et al. 2012], while multilayer perceptron (MLP) is chosen in the previous work [Ye and Xu 2012]. The output of an RBF network is

$$\sum_{k=1}^{K} \alpha_k \phi_k(\boldsymbol{I}), \tag{8}$$

where $\boldsymbol{I}$ is the input vector, $\phi_k$ is the $k$th RBF (hidden node) in the network with corresponding weight $\alpha_k$, and $K$ is the total number of hidden nodes in the network. Among many candidates, the Gaussian function is widely used as the RBF

$$\phi_k(\boldsymbol{I}) = \exp\left(-\frac{\|\boldsymbol{I} - \boldsymbol{u}_k\|^2}{\sigma_k^2}\right), \tag{9}$$

where $\boldsymbol{u}_k$ and $\sigma_k$ is the position vector and the width of the $k$th node, respectively, and $\|\cdot\|$ means the Euclidean norm.

Since the number of hidden nodes $K$ is hard to predefine to minimize approximation error without unnecessarily prolonging the runtime, the Resource Allocating Network

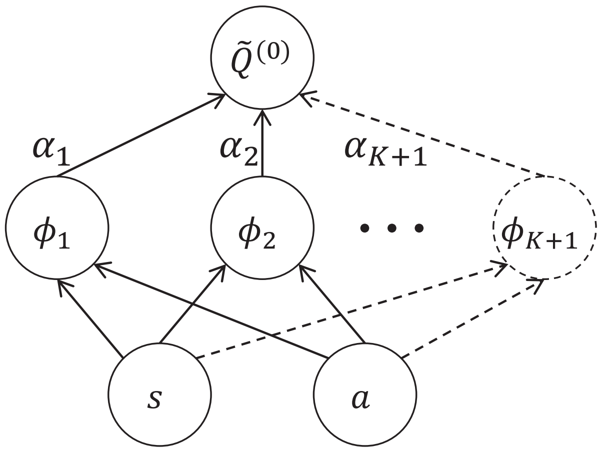

Fig. 4. Approximating Q-values by the Radial Basis Function (RBF) network.

(RAN) [Platt 1991] is exploited, that is based on the RBF network with the ability to dynamically allocate hidden nodes. When the output error $\Delta$ is larger than the desired accuracy $e_{min}$ but the current hidden nodes are all far from the input vector $\|\boldsymbol{I} - \boldsymbol{u}_k\| > \delta$ for all $k$, where $\delta$ is the threshold of distances between nodes, a new hidden node $\phi_{K+1}$ is allocated with

$$\begin{cases} \alpha_{K+1} \leftarrow \Delta \\ \boldsymbol{u}_{K+1} \leftarrow \boldsymbol{I} \\ \sigma_{K+1} \leftarrow \kappa \|\boldsymbol{I} - \boldsymbol{u}_k\| \end{cases}, \tag{10}$$

where $\kappa$ is the overlapping factor of hidden nodes. Otherwise, the network is adjusted by performing gradient descent

$$\omega \leftarrow \omega - \mu \frac{\partial \Delta}{\partial \omega}, \tag{11}$$

where $\omega$ is the target parameter for adjustment and $\mu$ the learning rate (the same as Q-learning); the weight $\alpha$ and the location $\boldsymbol{u}$ are adjusted in the RBF network by performing partial differentiation on the output error.

Furthermore, the Generalized Growing and Pruning RBF (GGAP-RBF) network [Huang et al. 2005] is able to eliminate insignificant hidden nodes in the RAN. The node nearest to the input vector is eliminated when its *significance* is smaller than the approximation accuracy $e_{min}$ and a new hidden node is allocated when it is *significant* enough for the input vector. Besides, the updating process (gradient descent) is restricted to the hidden node nearest to the input vector in order to speed-up the learning process.

To approximate the Q-values on the root node using the GGAP-RBF network, the input $\boldsymbol{I}$ is set to the normalized state-action pair ($\bar{s} = sq/mn, \bar{a} = a/n$) and each position vector $\boldsymbol{u_k}$ contains two dimensions $\boldsymbol{u_k} = (u_{sk}, u_{ak})$. The approximated Q-value on the root node is calculated by

$$\tilde{Q}^{(0)}(\bar{s}, \bar{a}) = \sum_{k=1}^{K} \alpha_k \exp\left(-\frac{(\bar{s} - u_{sk})^2 + (\bar{a} - u_{ak})^2}{\sigma_k^2}\right), \tag{12}$$

as shown in Figure 4. The updating of Q-values is similar to (5) where $\tilde{Q}^{(0)}(sq_{t-1}[0]/mn, sp_t[0]/n)$ is the approximated value and $(r_{t-1}[0] + \gamma \cdot \max_{0 \le a' \le n} \tilde{Q}^{(0)}(sp_t[0]/mn, a'/n))$

**ALGORITHM 2:** Update-Root

```
1  Update $(sp_{t-1}[0], sp_t[0], sq_{t-1}[0], sq_t[0], r_{t-1}[0])$ as the same as the coarsening phase;
2  $maxQ \leftarrow \max_{0 \le a' \le n} (\tilde{Q}^{(0)}(sq_t[0]/mn, a'/n))$;
3  $\Delta \leftarrow (r_{t-1}[0] + \gamma \cdot maxQ) - \tilde{Q}^{(0)}(sq_{t-1}[0]/mn, sp_t[0]/n))$;
4  $nr \leftarrow \arg\min_k \|(sq_{t-1}[0]/mn, sp_t[0]/n) - (u_{sk}, u_{ak})\|$;
5  $minD \leftarrow \|(sq_{t-1}[0]/mn, sp_t[0]/n) - \boldsymbol{u}_{nr}\|$;
6  if $minD > \delta$ and $|\Delta\kappa\sqrt{\pi/2} \cdot minD| > e_{min}$ then
7      Allocate new hidden node with $\boldsymbol{u}_{K+1} \leftarrow (sq_{t-1}[0]/mn, sp_t[0]/n)$, $\sigma_{K+1} \leftarrow \kappa \cdot minD$,
       $\alpha_{K+1} \leftarrow \Delta$;
8  end
9  else
10     Perform gradient descent on $\alpha_{nr}$;
11     if $|\alpha_{nr}\sigma_{nr}\sqrt{\pi/2}| > e_{min}$ then
12         remove $\phi_{nr}$ and the corresponding parameters $\boldsymbol{u}_{nr}$, $\sigma_{nr}$, $\alpha_{nr}$;
13     end
14 end
15 $\delta \leftarrow \max(\delta_{min}, \lambda\delta)$;
```

is the training value. Note that all the approximated Q-values of each state-action pair are changed on every single update, providing implicit interpolation for other Q-values.

The details of approximating $\tilde{Q}^{(0)}$ using the GGAP-RBF network are shown in Algorithm 2. The approximation error $\Delta$ and the minimum distance between the input vector and the hidden nodes *minD* are calculated in lines 2–5. If the distance is larger than the threshold $\delta$ and the error is significant against the desired accuracy $e_{min}$, a new hidden node is allocated in lines 6–8; otherwise, the network is adjusted by performing gradient descent on the nearest node $\phi_{nr}$ in line 10. Because the input vectors are discrete, the gradient descent adjustment is only performed on the weight $\alpha$ without moving the position. Then the significance of the nearest node $\phi_{nr}$ is checked in lines 11–13. The distance threshold $\delta$ is initialized to $\delta_{max}$ and then decays as time advances until $\delta_{min}$ with the decay constant $\lambda$ in line 15.

The most time-consuming operation in updating the RBF requires $n+1$ comparisons and each $\tilde{Q}^{(0)}$ requires $O(K)$ time to evaluate in line 2. Note that the number of hidden nodes $K$ is limited by the minimum distance parameter $\delta_{min}$ and the desired accuracy $e_{min}$ [Platt 1991; Huang et al. 2005]. In implementation, $K$ does not grow with $n$ because the input vector is normalized to the range [0, 1], thus the complexity is $O(n)$ for `Update-Root` using the GGAP-RBF network.

## 5.2. Smart Action Selection

Since the solution space is too large to try all of the state-action combinations, the agent faces the dilemma of exploration and exploitation: selecting the best action within the current knowledge, or trying other unknown actions? In the traditional $\epsilon$-greedy scheme, the trade-off parameter $\epsilon$ is a fixed value for all the states, which needs careful manual tuning. Hence, we choose another action selection scheme: the Value-Difference-based Exploration (VDBE)-softmax method [Tokic and Palm 2011], which outperforms other schemes when combined with Q-learning. The probability of jumping out the local optimum is calculated according to the value difference of the (approximated) Q-values; that is, the agent takes the greedy action when the system is stable and tries other actions when not. It is similar to $\epsilon$-greedy but replacing the

**ALGORITHM 3:** Decide-Root

```
1 $\epsilon(sq_{t-1}[0]) \leftarrow \frac{1}{n+1} \cdot \frac{1-\exp(-\mu|\Delta|)}{1+\exp(-\mu|\Delta|)} + (1 - \frac{1}{n+1}) \cdot \epsilon(sq_{t-1}[0]))$;
2 if $\xi < \epsilon(sq_t[0])$ then
3     $a_t[0] \leftarrow \arg\mathrm{softmax}_{0\le a' \le n}(\tilde{Q}^{(0)}(sq_t[0]/mn, a'/n))$;
4 end
5 else
6     $a_t[0] \leftarrow \arg\max_{0\le a' \le n}(\tilde{Q}^{(0)}(sq_t[0]/mn, a'/n))$;
7 end
```

uniform random selection by softmax probability

$$\frac{\exp{(Q(s,a)/\tau)}}{\sum_b \exp{(Q(s,b)/\tau)}}. \tag{13}$$

The temperature is kept constant $\tau = 1$ in VDBE-softmax so that other actions take higher probabilities to be tried, while the greedy action is taken most of the time with probability $1-\epsilon(s)$. Besides, the per-state threshold $\epsilon(s)$ is kept instead of using a global value, so the agent can explore different actions when the environment changes.

The details of VDBE-softmax are shown in Algorithm 3. The per-state exploration probability $\epsilon(s)$ is updated according to the value difference in the Boltzmann distribution where the learning rate is the inverse of the number of actions. The value of $\Delta$ is the same as in Algorithm 2; it is the difference between the target Q-value and the current (approximated) Q-value. When the system is unstable (at the beginning or when the environment changes), $\epsilon(s)$ is large so that the agent may explore other actions; otherwise, the agent takes the greedy choice to avoid jumping to other inferior states.

## 6. SIMULATION RESULTS

The proposed policy is compared with the previous work by simulation. First, the simulation environment and the parameters of the policies are described. Then, the policies are compared and analyzed in terms of energy saving and performance penalty using open-source benchmarks. Finally, the runtime overheads in executing the policies are analyzed. Because of the randomness in the simulator as well as the policies, all the results are averaged from at least ten runs.

### 6.1. Environment Settings

The multiprocessor performance simulator Multi2Sim [Ubal et al. 2007] and the power simulator McPAT [Li et al. 2009] are combined to support closed-loop dynamic power management. The transition overheads are implicitly modeled, including the hardware costs of mode switching, or refilling the pipeline or cache. The power manager is activated with period $T = 1$ms (between 0.5ms in Isci et al. [2006] and 10ms in Winter et al. [2010]). During each period, the performance statistics of the last period are collected from Multi2Sim and sent to McPAT to obtain the corresponding power statistics; both feedback information are sent to the power manager to determine the target power mode of the multiprocessor cores.

The target architecture is ARM Cortex-A9 MPCore [ARM 2012] where the configuration parameters are listed in Table II, obtained from its manual [ARM 2012] and McPAT [Li et al. 2009]. The technology parameters are supported by McPAT as well.

The workloads for simulations are the SPLASH-2 benchmarks [Woo et al. 1995] as listed in Table III, which are widely used to evaluate multiprocessor systems. The executables, arguments, and input files are obtained from Multi2Sim [Ubal et al. 2007].

Table II. Configuration Parameters of ARM Cortex A9 [ARM 2012]

| Parameter name | Parameter value |
|---|---|
| Number of cores | 4 |
| Number of threads per core | 1 |
| Technology node | 40nm |
| Operating frequency | 2000MHz |
| Supply voltage | 0.66V |
| Threshold voltage | 0.23V |
| Decode width | 2 |
| Issue width | 4 |
| Commit width | 4 |
| Number of ALUs per core | 3 |
| Number of MULs per core | 1 |
| Number of FPUs per core | 1 |
| Branch predictor | Two level, 1024-set 2-way BTB |
| L1 data cache | 32KB, 4 way, 10-cycle latency |
| L1 instruction cache | 32KB, 4 way, 10-cycle latency |
| L2 unified cache | 1MB, 8 way, 23-cycle latency |

Table III. SPLASH-2 Benchmarks

| Benchmark | Problem Size | Cycles (B) | Instructions (B) | Power (W) |
|---|---|---|---|---|
| Barnes | 2048 particles | 0.305 | 0.433 | 0.294 |
| Cholesky | tk14.O | 0.288 | 0.130 | 0.192 |
| FFT | 65536 points | 0.817 | 0.426 | 0.202 |
| FMM | 2048 particles | 0.319 | 0.542 | 0.333 |
| LU | $512 \times 512$ matrix, $16 \times 16$ blocks | 0.902 | 0.949 | 0.259 |
| Ocean | $130 \times 130$ ocean | 0.364 | 0.305 | 0.234 |
| Radiosity | -batch -en 0.5 | 0.940 | 1.104 | 0.271 |
| Radix | 256k keys, max-value 524288, radix 4096 | 0.154 | 0.227 | 0.298 |
| Raytrace | balls4 | 1.493 | 0.613 | 0.190 |
| Water-Nsq | 512 molecules, 1 timestep | 0.492 | 0.623 | 0.283 |
| Water-Sp | 512 molecules, 1 timestep | 0.421 | 0.550 | 0.287 |

The dynamic context scheduler is provided by Multi2Sim [Ubal et al. 2007]. Each program dynamically forks at most four parallel contexts during runtime. The context binding overheads are inherently modeled in the simulator.

Our policy (MLRL) is compared with the state-of-the-art work (BPNN) [Ye and Xu 2012]. The results are normalized to the baseline (BASE) without power management; the measured data of the baseline are shown in Table III. In addition, the oracle policy (GOLD) with maximum energy saving but zero performance penalty is listed as the reference upper bound. The parameters for Q-learning are set the same as in Ye and Xu [2012] with $\gamma = 0.5, \mu = 0.5$, and $\epsilon = 0.1$ for BPNN. The parameters for the GGAP-RBF network are set according to Platt [1991] and Huang et al. [2005] as $e_{min} = 0.05, \delta_{max} = 0.7, \delta_{min} = 0.07, \kappa = 0.87, \lambda = \exp(-1/17)$, and the inputs $(s, a)$ are normalized to [0, 1]. Both policies are initialized without workload knowledge or pretraining. For the reward calculated according to (2), the throughput is in the unit of instructions per cycle (IPC) and the power is in Watt (W).

### 6.2. Comparisons of Performance and Energy

*6.2.1. Analyses on Different Policies.* The statistics of running different policies with $\beta = 0.9$ for SPLASH-2 are shown in Table IV. The values of BPNN and MLRL are the normalized percentage compared with BASE. The average power saving of MLRL is

Table IV. Simulations of SPLASH-2 Benchmarks in $\beta = 0.9$

| Item (%) | Power Saving | | | Perf. Penalty | | | Energy Saving | | |
|---|---|---|---|---|---|---|---|---|---|
| Benchmark | BPNN | MLRL | GOLD | BPNN | MLRL | GOLD | BPNN | MLRL | GOLD |
| Barnes | 15.09 | 2.81 | 2.73 | 17.78 | 1.25 | 0.00 | 0.01 | 1.61 | 2.73 |
| Cholesky | 26.92 | 40.14 | 43.64 | 9.08 | 0.86 | 0.00 | 20.31 | 39.65 | 43.64 |
| FFT | 24.72 | 28.07 | 25.25 | 11.77 | 5.93 | 0.00 | 15.85 | 23.80 | 25.25 |
| FMM | 9.43 | 5.65 | 3.23 | 10.29 | 3.74 | 0.00 | 0.11 | 2.11 | 3.23 |
| LU | 14.85 | 14.63 | 11.98 | 13.79 | 3.96 | 0.00 | 3.10 | 11.24 | 11.98 |
| Ocean | 30.60 | 4.72 | 0.00 | 42.61 | 8.03 | 0.00 | −9.68 | −2.34 | 0.00 |
| Radiosity | 10.90 | 8.81 | 8.91 | 6.90 | 0.01 | 0.00 | 4.76 | 8.83 | 8.91 |
| Radix | 8.77 | 4.06 | 0.00 | 10.06 | 4.19 | 0.00 | −0.50 | −0.08 | 0.00 |
| Raytrace | 24.15 | 49.54 | 49.72 | 2.40 | 0.02 | 0.00 | 22.34 | 49.54 | 49.72 |
| Water-Nsq | 21.37 | 8.54 | 8.34 | 22.49 | 0.72 | 0.00 | 3.69 | 7.89 | 8.34 |
| Water-Sp | 18.63 | 7.60 | 7.48 | 18.72 | 0.59 | 0.00 | 2.85 | 7.07 | 7.48 |
| Average | 18.63 | 15.87 | 14.66 | 15.08 | 2.66 | 0.00 | 5.71 | 13.58 | 14.66 |

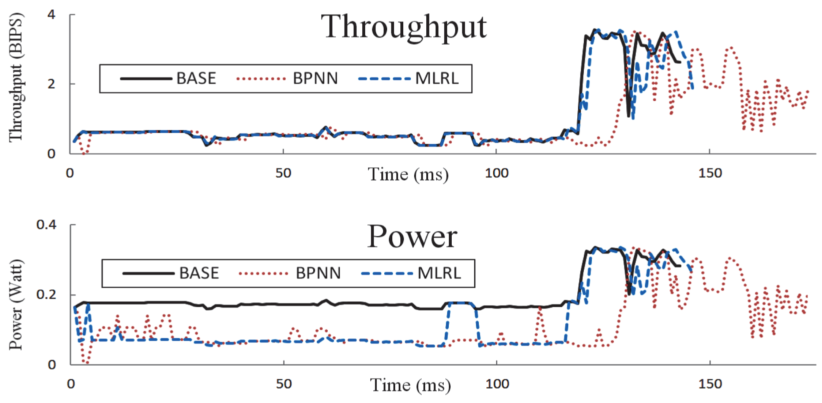

Fig. 5. Transient results of Cholesky with $\beta = 0.9$.

close to BPNN, but the performance penalty is much lower, resulting in much more energy saving. The main reason is that the rapid convergence rate of MLRL avoids trying lots of inferior actions. Moreover, VDBE-softmax ensures the optimal action is selected after convergence, while the $\epsilon$-greedy occasionally jumps to other states. For MLRL, the energy saving is 13.58% on average and up to 49.54%, while the performance penalty is 2.66% on average and within 8.03% at worst.

For some extreme cases (Barnes, FMM, Ocean, Radix), both policies achieve low or even negative energy saving. Because the cores are busy almost all the time, turning off cores cannot save energy but can lengthen the latency. Furthermore, some overheads are induced when the cores switch, such as filling the pipelines and binding (or ejecting) contexts to cores. In these cases, the performance penalty and energy saving of MLRL is better than BPNN, because our policy with VDBE-softmax seldom erroneously turns off busy cores.

On the contrary, some programs (Cholesky, FFT, Raytrace) run serially for a long time, yielding lots of idle periods for most cores. As the transient diagrams of Cholesky shown in Figure 5 show, the fast converging for MLRL at the beginning (0–40ms) results in higher energy saving and the rapid reaction to the environment changes (120–130ms) leads to a shorter latency penalty.

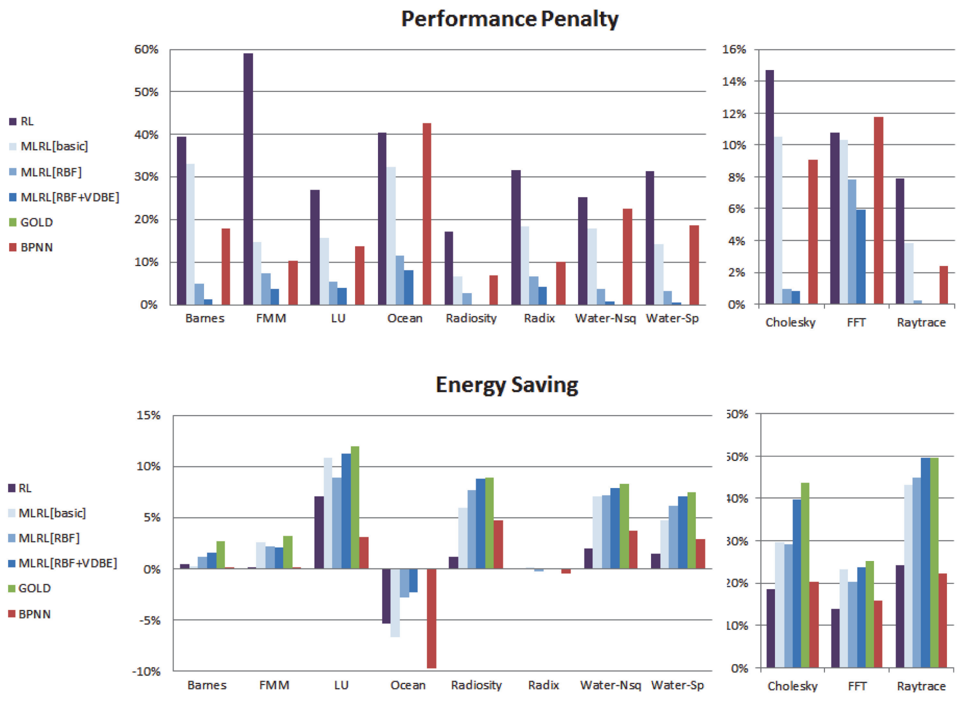

Fig. 6. The comparisons on performance penalty and energy saving of different techniques.

*6.2.2. Analyses on the Enhancement Techniques.* The performance penalty and energy saving with different enhancement techniques described in Section 5 are shown in Figure 6 for analysis. Six policies are compared: the original reinforcement learning policy (RL), MLRL without any enhancement technique (MLRL[basic]), MLRL with GGAP-RBF without VDBE-softmax (MLRL[RBF]), MLRL with all the enhancement techniques (MLRL[RBF+VDBE]), the golden reference (GOLD), and the BPNN policy (BPNN). In order to show the figures clearly, the three benchmarks (Cholesky, FFT, Raytrace) with much larger energy saving are shown in different scales.

The RL policy that directly encodes the power states results in the largest performance penalty but least energy saving due to its large state space (at least $2^n$). For MLRL[basic], the performance penalty is reduced but still huge, because the state space is still large on the root node; many samples are needed for $n$ states and $n$ actions, but the agent seldom tries other actions using $\epsilon$-greedy. The average performance penalty for MLRL[basic] (16.14%) is comparable to BPNN (15.08) with higher energy saving ($10.99\% > 5.71\%$), demonstrating the proposed multilevel paradigm is more effective than directly applying function approximation with BPNN.

Then the function approximation MLRL[RBF] greatly reduces the average performance penalty ($16.14\% \rightarrow 4.98\%$), because the agent does not need to visit all the solutions with sufficient samples to select appropriate actions. The energy saving is similar to MLRL[basic] ($10.99\% \rightarrow 11.31\%$) and close to the upper bound.

Finally, including the VDBE-softmax scheme further reduces the average performance penalty ($4.98\% \rightarrow 2.66\%$) and saves more energy ($11.31\% \rightarrow 13.58\%$). The performance penalty of MLRL[RBF+VDBE] is close to zero, while the energy saving is close to GOLD (14.66%). The difference (about 1% on average) is due to the

Table V. Simulations of SPLASH-2 Benchmarks in $\beta = 0.9$ with Three Cores

| Item (%) | Power Saving | | | Perf. Penalty | | | Energy Saving | | |
|---|---|---|---|---|---|---|---|---|---|
| Benchmark | BPNN | MLRL | GOLD | BPNN | MLRL | GOLD | BPNN | MLRL | GOLD |
| Barnes | 8.11 | 2.76 | 1.43 | 8.76 | 1.57 | 0.00 | 0.06 | 1.23 | 1.43 |
| Cholesky | 17.77 | 33.79 | 34.86 | 5.09 | 1.01 | 0.00 | 13.59 | 33.12 | 34.86 |
| FFT | – | – | – | – | – | – | – | – | – |
| FMM | 7.57 | 4.25 | 1.49 | 8.18 | 3.16 | 0.00 | 0.01 | 1.23 | 1.49 |
| LU | 3.64 | 10.07 | 9.02 | 3.24 | 2.10 | 0.00 | 0.52 | 8.18 | 9.02 |
| Ocean | – | – | – | – | – | – | – | – | – |
| Radiosity | 6.61 | 6.10 | 6.18 | 4.87 | 0.02 | 0.00 | 2.05 | 6.13 | 6.18 |
| Radix | – | – | – | – | – | – | – | – | – |
| Raytrace | 22.19 | 40.04 | 40.06 | 2.72 | 0.22 | 0.00 | 20.08 | 39.90 | 40.06 |
| Water-Nsq | 9.16 | 6.32 | 6.06 | 5.22 | 0.50 | 0.00 | 4.42 | 5.85 | 6.06 |
| Water-Sp | 13.79 | 20.69 | 20.33 | 4.37 | 1.01 | 0.00 | 10.02 | 19.89 | 20.33 |
| Average | 11.11 | 15.50 | 14.93 | 5.31 | 1.19 | 0.00 | 6.35 | 14.44 | 14.93 |

Table VI. Simulations in Different Trade-Off Parameters ($\beta$)

| Item (%) | Power Saving | | Perf. Penalty | | Energy Saving | |
|---|---|---|---|---|---|---|
| $\beta$ | BPNN | MLRL | BPNN | MLRL | BPNN | MLRL |
| 0.1 | 40.43 | 37.47 | 98.21 | 63.78 | 9.79 | 10.73 |
| 0.3 | 33.64 | 29.69 | 27.57 | 35.76 | 8.29 | 13.65 |
| 0.5 | 19.69 | 21.29 | 19.62 | 12.85 | 5.60 | 13.08 |
| 0.7 | 19.26 | 17.72 | 15.93 | 4.27 | 7.03 | 13.94 |
| 0.9 | 18.63 | 15.87 | 15.08 | 2.66 | 5.71 | 13.58 |

nature of online learning where the agent still needs to try some actions before convergence.

*6.2.3. Non-Power-of-Two Architectures.* In order to examine whether the proposed policy can be applied to general architectures, the ARM MPCore platform is equipped with three cores. Similar to Table IV, the statistics with $\beta = 0.9$ are shown in Table V where the values of BPNN and MLRL are the improvement percentage compared with BASE. Some benchmarks (FFT, Ocean, and Radix) are skipped here because they can only be run on power-of-two architectures. For MLRL, the average energy saving (14.44%) is very close to the upper bound (14.93%), while the average performance penalty is only 1.19%. MLRL still works effectively for three cores and outperforms BPNN in either energy saving and performance penalty, showing the proposed policy is general and applicable for multiprocessor systems with non-power-of-two cores.

Compared with four cores, the room for energy saving is lower (GOLD: 17.00% $\rightarrow$ 14.93% without FFT, Ocean, and Radix) because the number of cores that can be turned off is reduced by one when the application runs in serial segments (Water-Spatial is an exception where the program leaves more idle cores on the three-core system). As a result, the power saving is lower for both policies (BPNN: 17.61% $\rightarrow$ 11.11%, MLRL: 17.22% $\rightarrow$ 15.50%). The performance penalty is also lower for both policies (BPNN: 12.68% $\rightarrow$ 5.31%, MLRL: 1.39% $\rightarrow$ 1.19%) because the solution spaces are reduced due to smaller $n$ so that the agents can more quickly find adequate solutions.

*6.2.4. Analyses on Different Values of the Trade-Off Parameter.* The simulation results of varying the trade-off parameter $\beta$ are shown in Table VI. All these numbers are averaged across the 11 benchmarks and normalized to BASE.

For smaller $\beta$, the power saving is greater but the performance penalty is higher for both policies, as expected. The energy saving is relatively stable with respect to $\beta$ because the *true energy* to complete execution cannot be saved. The energy saving of

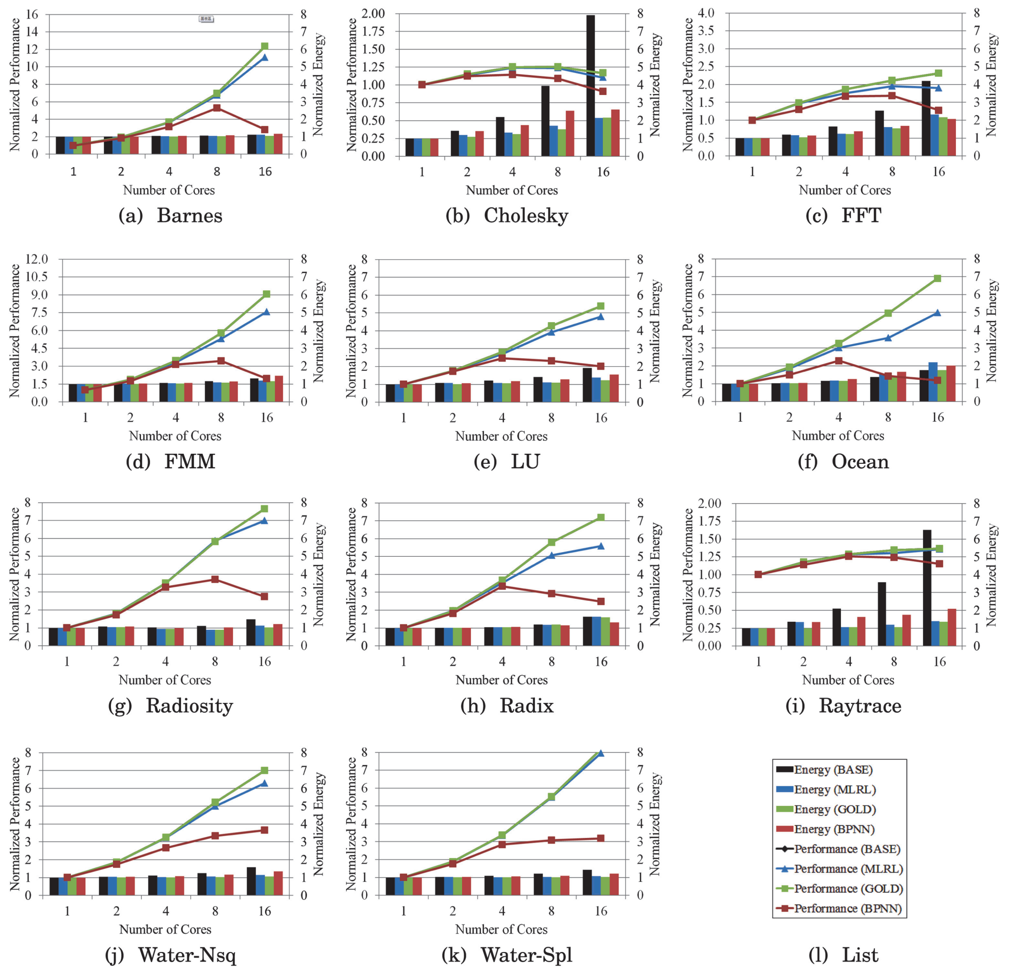

Fig. 7. The scalability in performance and energy consumption of different policies for SPLASH-2 benchmarks.

MLRL is greater than BPNN for any value of $\beta$, showing the superiority and stability of our policy. Because the energy saving is stable while the performance penalty is lower, higher $\beta$ values are suitable for MLRL.

Varying the other two parameters $\mu$ and $\gamma$ in Q-learning does not affect MLRL significantly, hence the results are not shown. The parameters $e_{min}$, $\delta$, $\lambda$ and $\kappa$ in GGAP-RBF are mutually related, which cannot be changed arbitrarily but must be set to the original values.

*6.2.5. Analyses on the Scalability of Different Policies.* In order to compare the scalability of different policies, the architecture is scaled to more cores. Note that the architectures with more than four cores were not real systems (for ARM Cortex A9) at the time when this work was submitted; they are virtually created only to analyze the scalability of the proposed MLRL policy.

The scalability diagrams are depicted in Figure 7 for different benchmarks. The performance is normalized to uniprocessors, thus the lines are the speedup of different policies with varying number of cores. The energy consumption shown in the bar

Table VII. Runtime Analysis

| Benchmark | BPNN ($\mu s$) | MLRL ($\mu s$) | Improvement |
|---|---|---|---|
| Barnes | 28.96 | 15.42 | 47% |
| Cholesky | 19.27 | 8.22 | 57% |
| FFT | 16.89 | 10.77 | 36% |
| FMM | 29.12 | 15.67 | 46% |
| LU | 23.24 | 11.73 | 50% |
| Ocean | 27.23 | 14.31 | 47% |
| Radiosity | 13.93 | 10.56 | 24% |
| Radix | 29.34 | 8.80 | 70% |
| Raytrace | 23.00 | 5.68 | 75% |
| Water-Nsq | 26.10 | 9.73 | 63% |
| Water-Sp | 27.68 | 8.87 | 68% |
| Average | 24.07 | 10.89 | 53% |

chart is normalized to uniprocessors as well. In order to show the speedup clearly, the scales of the normalized performance are different; the benchmarks Cholesky, FFT, and Raytrace that contain long serial parts result in poor scalability in performance and large room for energy saving.

In general, MLRL is scalable in performance speedup and energy saving, outperforming BPNN for eight or sixteen cores. Since the number of epochs decreases while the searching space becomes larger for more cores, the performance and energy gaps with respect to GOLD are accordingly widened for both MLRL and BPNN. For Cholesky, which has the poorest scalability [Woo et al. 1995], the speedup goes downward for eight and sixteen cores even for GOLD due to the resource (cache) contention problem in our target embedded architectures (the memory subsystem in Woo et al. [1995] is assumed perfect). The performance speedup values of MLRL decrease for FFT and Raytrace because the trade-off parameter $\beta$ in (2) is constant; when the performance speedup faces diminishing returns while the energy consumption becomes larger for more cores, MLRL would choose to turn off more cores to save energy.

## 6.3. Comparisons of Policy Overhead

Finally, the runtime overheads of executing the policies are evaluated and compared, while the transition overheads between power modes are implicitly included in the previous performance analysis. Because our instruction-set simulator Multi2Sim [Ubal et al. 2007] does not contain a full operating system, the policy runtime is measured separately using the statistics from previous sections as inputs. The runtime analysis is conducted on an Intel Xeon CPU E5420 running at 2.5 GHz using the `gettimeofday` function.

*6.3.1. Overhead Comparisons.* The runtime comparisons of different policies are shown in Table VII with $\beta = 0.9$. For all of the benchmarks, MLRL is faster than BPNN with 53% improvement on average, showing the efficiency of the proposed policy. Note that both of the policies are fast enough (in $\mu s$) for four cores.

Among the different benchmarks, the runtime variation of MLRL mainly comes from the (dynamic) number of hidden nodes in the GGAP-RBF architecture. Taking Cholesky as an example, the input state $\bar{s}$ remains $1/n$ for a long time at the beginning due to the serial segment of the program, so the GGAP-RBF network contains only a few hidden nodes. The same situation occurs for FFT and Raytrace, while the GGAP-RBF network is more complex for Barnes, FMM, and Ocean where the parallelism of workloads varies rapidly.

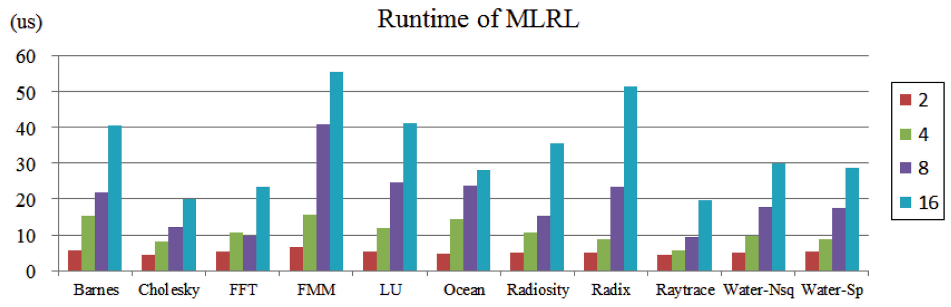

Fig. 8. The runtime of the proposed policy with varying the number of cores for different benchmarks.

*6.3.2. Analysis on Scalability.* Then the number of cores is scaled up to examine the complexity of the proposed policy. The number of cores and the parallelism of benchmarks are set to two, four, eight, and sixteen.

The runtime of MLRL is shown in Figure 8 with differing number of cores $n$ for the eleven SPLASH-2 benchmarks. Although the runtime is affected by the $O(Kn)$ time in updating the GGAP-RBF network, the number of hidden nodes $K$ is saturated due to the constant desired accuracy $e_{min}$ and the minimum distance $\delta_{min}$ in the normalized space; the resulting number of hidden nodes is around 6–8 on average, regardless of the number of cores $n$. Therefore, the overall runtime is dominated by $O(n \lg n)$ as analyzed in the previous sections.

## 7. RELATED WORKS

Several power management policies have been proposed in the past decades [Benini et al. 2000]; they can be classified into four categories, namely timeout, predictive, stochastic, and learning based. The timeout policies [Karlin et al. 1994; Golding et al. 1996] are simple, but the waiting time means wasting of power. The predictive [Hwang and Wu 2000; Augustine et al. 2008] and stochastic policies [Qiu et al. 2007; Jung and Pedram 2009] improve this drawback using heuristics- and model-based approaches, respectively; however, these policies either require some preknowledge or assume task models. Machine learning techniques have been widely applied recently [Dhiman and Simunic Rosing 2009; Tan et al. 2009] that consider the variability of working environments and outperform previous approaches.

The aforesaid policies designed for uniprocessor systems can be applied to multiprocessors using either chip-wide DPM (treating the whole multiprocessor system as a core) [Kveton et al. 2007] or distributed power management (treating each core as an independent uniprocessor) [Shen et al. 2013]. They are both less effective than centralized policies because the former approach cannot switch in per-core granularity while the latter method lacks of a global view. In Madan et al. [2011], two static heuristics are proposed for datacenters, but the power manager is turned off when it encounters problems. In Mariani et al. [2011], an application-specific framework is proposed with design-time characterization of applications.

For multiprocessors, other types of policy exploit Dynamic Voltage and Frequency Scaling (DVFS) [Maggio et al. 2012], such as MaxBIPS [Isci et al. 2006], Steepest Drop [Winter et al. 2010], supervised learning [Jung and Pedram 2010], and hierarchical gradient ascent [Sartori and Kumar 2009]. The Thread Motion (TM) technique [Rangan et al. 2009] also exploits MaxBIPS and migrates applications between cores. The main considerations of DVFS are transient power and thermal issues, while the focus of DPM is overall energy saving. Moreover, the transition overheads that are ignored in most DVFS policies should be taken into consideration in DPM. Because DPM and

DVFS outperform each other in some situations, they can be designed separately and then combined to provide comprehensive power management by static characterization [Srivastav et al. 2012], utilizing different states and rewards for the learning agent [Shen et al. 2013], or policy(expert) selection [Dhiman and Simunic Rosing 2009; Bhatti et al. 2010].

## 8. CONCLUSION

In this article, we propose a learning-based power management policy called MLRL for multiprocessors. Using the multilevel paradigm, the time complexity is reduced to $O(n \lg n)$ and the searching space is reduced to linearly proportional to the number of cores. Moreover, the scalability is further enhanced using function approximation by the GGAP-RBF network and the convergence quality is raised by using the VDBE-softmax action selection technique. The hierarchical approach is generalized to multiprocessor systems with non-power-of-two cores.

The simulations are conducted on the instruction-set simulator using the SPLASH-2 benchmarks. The results show that MLRL requires shorter runtime and outperforms the state-of-the-art policy. MLRL runs 53% faster and achieves 13.6% energy saving with only 2.7% performance penalty on average. The energy saving of MLRL is close to the oracle policy with only 1.1% difference on average. The effects of the enhancement techniques are evaluated as well. In addition, the generality and scalability of the proposed policy are examined on architectures with two to sixteen cores.

In the future, MLRL can be extended to DVFS or even single-ISA heterogeneous systems by generalizing the definitions of state and action in the multilevel framework such that the number of active cores becomes the aggregate computational capability. The heterogeneity and correlation between different contexts can be further taken into consideration where the power manager can be combined with the task scheduler to provide a comprehensive policy for complicated workloads. Moreover, the power consumption or performance penalty can be controlled by adding a controller on the value of $\beta$.

## REFERENCES

ACPI. 2011. ACPI - Advanced configuration and power interface specification. http://www.acpi.info/.

AMD. 2013. AMD powernow! technology. http://www.amd.com/us/products/technologies/amd-powernow-technology/Pages/amd-powernow-technology.aspx.

ARM. 2005. ARM intelligent energy controller technical overview. http://www.arm.com/.

ARM. 2012. Cortex-a9 mpcore technical reference manual. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0407i/DDI0407I_cortex_a9_mpcore_r4p1_trm.pdf.

John Augustine, Sandy Irani, and Chaitanya Swamy. 2008. Optimal power-down strategies. *SIAM J. Comput.* 37, 5, 1499–1516.

Andrew G. Barto and Sridhar Mahadevan. 2003. Recent advances in hierarchical reinforcement learning. *Discr. Event Dynam. Syst.* 13, 1–2, 41–77.

Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. 2000. A survey of design techniques for system-level dynamic power management. *IEEE Trans. VLSI Syst.* 8, 3, 299–316.

Khurram Bhatti, Cecile Belleudy, and Michel Auguin. 2010. Power management in real time embedded systems through online and adaptive interplay of dpm and dvfs policies. In *Proceedings of the IEEE/IFIP International Conference on Embedded and Ubiquitous Computing.*

Gaurav Dhiman and Tajana Simunic Rosing. 2009. System-level power management using online learning. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst.* 28, 5, 676–689.

Richard Golding, Peter Bosch, Carl Staelin, Tim Sullivan, and John Wilkes. 1996. Idleness is not sloth. Tech. rep. Hewlett-Packard Laboratories, Palo Alto, CA.

Jim Held, Jerry Bautista, and Sean Koehi. 2006. From a few cores to many: A tera-scale computing research overview. Tech. rep., Intel Corporation.

Guang-Bin Huang, P. Saratchandran, and Narasimhan Sundararajan. 2005. A generalized growing and pruning rbf (ggap-rbf) neural network for function approximation. *IEEE Trans. Neural Netw.* 16, 1, 57–67.

Chi-Hong Hwang and Allen C.-H. Wu. 2000. A predictive system shutdown method for energy saving of event-driven computation. *ACM Trans. Des. Autom. Electron. Syst.* 5, 2, 226–241.

Intel. 2013. Enhanced intel speedstep technology. http://www3.intel.com/cd/channel/reseller/asmo-na/eng/203838.htm.

Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. 2006. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*.

Niraj K. Jha. 2001. Low power system scheduling and synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design.*

Hwisung Jung and Massoud Pedram. 2009. Uncertainty-aware dynamic power management in partially observable domains. *IEEE Trans. VLSI Syst.* 17, 7, 929–942.

Hwisung Jung and Massoud Pedram. 2010. Supervised learning based power management for multicore processors. *IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst.* 29, 9, 1395–1408.

Anna R. Karlin, Mark S. Manasse, Lyle A. Mcgeoch, and Susan Owicki. 1994. Competitive randomized algorithms for non-uniform problems. *Algorithmica* 11, 6, 542–571.

George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. 1999. Multilevel hypergraph partitioning: Application in vlsi domain. *IEEE Trans. VLSI Syst.* 7, 1, 69–79.

Wonyoung Kim, Meeta S. Gupta, Gu-Yeon Wei, and David Brooks. 2008. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *Proceedings of the 14th IEEE International Symposium on High Performance Computer Architecture*. 123–134.

Matthias Knoth. 2009. Power management in an embedded multiprocessor cluster. In *Proceedings of the Embedded World Conference.*

Branislav Kveton, Prashant Gandhi, Georgios Theocharous, Shie Mannor, Barbara Rosario, and Nilesh Shah. 2007. Adaptive timeout policies for fast fine-grained power management. In *Proceedings of the 19th National Conference on Innovative Applications of Artificial Intelligence.*

Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture.*

Niti Madan, Alper Buyuktosunoglu, Pradip Bose, and Murali Annavaram. 2011. A case for guarded power gating for multi-core processors. In *Proceedings of the International Symposium on High-Performance Computer Architecture.*

Martina Maggio, Henry Hoffmann, Alessandro V. Papadopoulos, Jacopo Panerati, Marco D. Santambrogio, Anant Agarwal, and Alberto Leva. 2012. Comparison of decision-making strategies for self-optimization in autonomic computing systems. *ACM Trans. Auton. Adapt. Syst.* 7, 4, 36:1–36:32.

Giovanni Mariani, Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. 2011. ARTE: An application-specific run-time management framework for multi-core systems. In *Proceedings of the IEEE Symposium on Application Specific Processors.*

Massoud Pedram. 1996. Power minimization in ic design: Principles and applications. *ACM Trans. Des. Autom. Electron. Syst.* 1, 1, 3–56.

John Platt. 1991. A resource-allocating network for function interpolation. *Neural Comput.* 3, 2, 213–225.

Qinru Qiu, Ying Tan, and Qing Wu. 2007. Stochastic modeling and optimization for robust power management in a partially observable system. In *Proceedings of the Design, Automation, and Test in Europe Conference*. 779–784.

Krishna K. Rangan, Gu-Yeon Wei, and David Brooks. 2009. Thread motion: Fine-grained power management for multi-core systems. In *Proceedings of the International Symposium on Computer Architecture.*

John Sartori and Rakesh Kumar. 2009. Distributed peak power management for many-core architectures. In *Proceedings of the Design, Automation, and Test in Europe Conference.*

Joseph Sharkey, Alper Buyuktosunoglu, and Pradip Bose. 2007. Evaluating design tradeoffs in on-chip power management for cmps. In *Proceedings of the International Symposium on Low Power Electronics and Design.*

Hao Shen, Ying Tan, Jun Lu, Qing Wu, and Qinru Qiu. 2013. Achieving autonomous power management using reinforcement learning. *ACM Trans. Des. Autom. Electron. Syst.* 18, 2.

Meeta Srivastav, Michael B. Henry, and Leyla Nazhandali. 2012. Design of energy-efficient, adaptable throughput systems at near/sub-threshold voltage. *ACM Trans. Des. Autom. Electron. Syst.* 18, 1.

Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. The MIT Press.

Ying Tan, Wei Liu, and Qinru Qiu. 2009. Adaptive power management using reinforcement learning. In *Proceedings of the International Conference on Computer Aided Design.*

Chen Khong Tham. 1994. Modular on-line function approximation for scaling up reinforcement learning. Ph.D. dissertation, Jesus College, Cambridge, UK.

Michel Tokic and Gunther Palm. 2011. Value-difference based exploration: Adaptive control between epsilon-greedy and softmax. In *Proceedings of the 34th Annual German Conference on Advances in Artificial Intelligence.*

Rafael Ubal, Julio Sahuquillo, Salvador Petit, and Pedro Lopez. 2007. Multi2Sim: A simulation framework to evaluate multicore-multithreaded processors. In *Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing*.

Jonathan A. Winter, David H. Albonesi, and Christine A. Shoemaker. 2010. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques.*

Steven C. Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder P. Singh, and Anoop Gupta. 1995. The splash-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture.*

Yue Wu, Hui Wang, Biaobiao Zhang, and Ke-Lin Du. 2012. Using radial basis function networks for function approximation and classification. *ISRN Appl. Math. 2012*.

Rong Ye and Qiang Xu. 2012. Learning-based power management for multi-core processors via idle period manipulation. In *Proceedings of the Asia and South Pacific Design Automation Conference*.

Received July 2013; revised December 2013, March 2014; accepted April 2014