Improving the Efficiency of Power Management Techniques by Using Bayesian Classification[1]

Hwisung Jung and Massoud Pedram
Department of Electrical Engineering, University of Southern California
Los Angeles CA, USA
{*hwijung, pedram*}*@usc.edu*

Abstract.

This paper presents a supervised learning based dynamic power management (DPM) framework for a multicore processor, where a power manager (PM) learns to predict the system performance state from some readily available input features (such as the state of service queue occupancy and the task arrival rate) and then uses this predicted state to look up the optimal power management action from a pre-computed policy lookup table. The motivation for utilizing supervised learning in the form of a Bayesian classifier is to reduce overhead of the PM which has to recurrently determine and issue voltage-frequency setting commands to each processor core in the system. Experimental results reveal that the proposed Bayesian classification based DPM technique ensures system-wide energy savings under rapidly and widely varying workloads.

1. Introduction

Ongoing advances in CMOS process technologies and VLSI designs have resulted in the introduction of multicore processors. There is, however, the need to achieve high system-level performance without driving up the power dissipation and chip temperature. Conventional dynamic power management (DPM) methods have not been able to take full advantage of power-saving solutions such as dynamic voltage and frequency scaling (DVFS). This is because a system-level power management routine, which continuously monitors the workloads of multiple processors, analyzes the information to make decisions, and issues DVFS commands to each processor, can add computational overhead and/or complicate the task scheduling [1]. Furthermore, the ability of a DPM method to scale well on a multicore processor by eliminating these overheads is becoming a critical requirement [2][3].

The problem of determining a power management policy exploiting DVFS in a multicore processor or a network-on-chip has received a lot of attention. In [4], the authors propose an online DVFS technique by utilizing an interface queue to guide the DVFS control in multiple clock and voltage domain architecture. Analytical models for system-level power management under tight performance constraints are presented in [5]. The work in [6] presents a voltage island-based power management technique to satisfy the required performance in multi-threshold CMOS technologies. The authors of [7] implement a power-efficient network-on-chip with a packet-switched serial-communication infrastructure, while integrating multiple IP cores. Learning-based techniques proposed in [8][9] use adaptive control mechanisms to select an optimal power management policy for embedded systems.

Although above-mentioned techniques [4]-[7] perform system-level dynamic power management or DVFS technique for multicore processors, little attention has been paid to improve decision-making strategy which minimizes the system overhead of a power manager (PM), i.e., to devise a learning-based power management policy that can quickly analyze some easily available input features and accurately predict the overall system state, which is subsequently used to select the "optimal action". For example, traditional approaches for DPM, based on models of service requestor (SR), service provider (SP), and service queue (SQ), work well only in the case of that the workload of the system does not change very rapidly. Otherwise, the energy and delay overhead of the power mode transitions can become quite significant, rendering the DPM strategy ineffective. Indeed, adaptive power management techniques [8][9] are unsuccessful in reducing the total chip power when the overhead of power-mode transitions is not controlled in multicore processors, where the PM needs to control each processor individually.

In this paper, we will address a system-level power management problem where a PM continually issues mode transition commands to maximally exploit the power-saving opportunities. The overhead associated with regular activity of the PM to monitor the workload of the system and make decisions about performance mode (voltage and frequency level) of different functional blocks in the system tends to be high. This paper thus describes a supervised learning [10] based DPM framework for a multicore processor, which enables the PM to predict the performance state of the system for each incoming task by a simple and efficient analysis of some readily available input features. Experimental results demonstrate the effectiveness of the framework and show that the proposed DPM technique ensures system-wide energy savings under rapidly varying workloads.

The remainder of this paper is organized as follows. Section 2 provides a motivational example. The details of the proposed power management framework are given in section 3. Experimental results and conclusions are given in section 4 and section 5.

2. Motivational Example

Considering conventional DPM approaches in a multicore processor, where each processor core is equipped with multiple power-saving modes (i.e., different DVFS sets), a system-level PM issues an optimal DVFS value to each processor based on the current workload by monitoring the flow queue as shown in Figure 1. This figure shows an example of multicore processor architecture, so-called distributed shared-memory multicore processor [11], where the dynamic load balancing provides high-throughput and low-latency data flow for each processor, and the control unit supports high-performance and power-efficient cache coherency. Details of the processor functionality are omitted to save space. Interested readers may refer to [11]. To capture power-saving opportunities inside the multicore processor, we consider the power consumption of each processor, controlled by the PM with various DVFS sets.

[1] This research is supported in part by the National Science Foundation under a CSR-EHS grant with award no. 0509564.

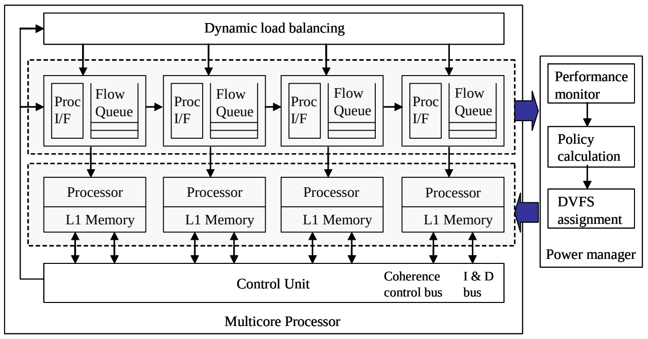

Figure 1. Example of multicore processor architecture.

Assume that when jobs are given to the multicore processor, the dynamic load balancing block (i.e., SR) writes the tasks into each flow queue (i.e., SQ), while each processor (i.e., SP) reads the tasks from its corresponding SQ. The PM monitors the workload of the processor by looking at the corresponding SQ in order to assign the DVFS value for the processor at each (regular or interrupt-driven) event occurrence, called *decision epochs*. Note that the decision epochs of the PM are separated by a time step, where the shorter this time step is, the higher the delay and energy dissipation penalties are. It is because that the DVFS method based on a voltage regulator and a PLL incurs non-negligible power-mode transition delay and energy dissipation [12]. Thus, the traditional DPM approach may result in sizeable performance penalty as shown in Figure 2(a).

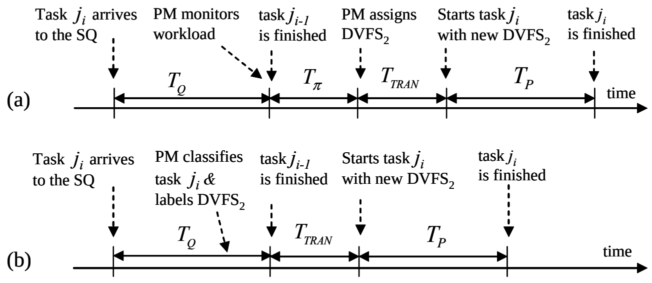
Task j_i arrives to the SQ
PM monitors workload
task j_{i-1} is finished
PM assigns DVFS$_2$
Starts task j_i with new DVFS$_2$
task j_i is finished
T_Q
T_π
T_{TRAN}
T_P
time
(a)
Task j_i arrives to the SQ
PM classifies task j_i & labels DVFS$_2$
task j_{i-1} is finished
Starts task j_i with new DVFS$_2$
task j_i is finished
T_Q
T_{TRAN}
T_P
time
(b)

Figure 2. DPM approaches with DVFS technique: (a) the traditional approach. (b) the proposed approach.

In this scenario (see Figure 2(a)), where T_Q is the queuing time (i.e., the time spent by a task in the SQ), and T_P is the time interval between two consecutive read operations at the SQ by the SP, we assume that task j_{i-1} is running optimally (i.e., achieves the required performance level with minimal amount of energy) with DVFS$_1$. Similarly, task j_i runs optimally under DVFS$_2$, where DVFS$_1$ < DVFS$_2$ in terms of voltage and frequency values. Then, the procedure for traditional DPM is explained as follows. First, task j_i arrives to the SQ and waits for time T_Q before reaching the SP. During T_Q, the PM monitors the workload of the SQ and calculates the optimal power management command, which takes T_π time. This time increases proportionally with the number of SPs, and is highly dependent on the speed of the operating system. After making the DVFS selections, the PM enables the voltage regulator or the PLL to adjust the supply voltage or operating frequency for the SP, which take T_{TRAN} time (= max (τ_{TRAN}, τ_{PLL})), where τ_{TRAN} is the transition time of voltage regulator, and τ_{PLL} is the PLL lock time. Finally, the SP takes T_P time to complete task j_i. The problems with this approach are: i) when the workload changes, each SQ (or controller) has to send an interrupt to the PM to request DVFS adjustments for the corresponding processor, which significantly increases the computational overhead of the PM as the number of processors increases, and ii) if the PM monitors every SQ to decide the DVFS values at the decision epochs, it has to schedule a series of DVFS assignments for every processor since all of tasks are not completed at the same time, which adds another scheduling overhead.

The main contribution of our work is that the incoming tasks are labeled with the system state information (power dissipation and execution time) and DVFS values are assigned to these tasks before they reach the SP. This saves the required time of monitoring the workload and generating an interrupt, as shown in Figure 2(b).

3. Learning-based DPM Framework

In this section, we present a theoretical framework to construct a supervised learning-based dynamic power management framework.

3.1 Background on Supervised Learning

Supervised learning [10] is an effective and practical technique for discovering relations and extracting knowledge in cases where the mathematical model of the problem may be too expensive to construct, or not available at all. Alternatively, it may be used to derive a *self-improving* decision-making strategy instead of making decisions based on the current perceived state of the system.

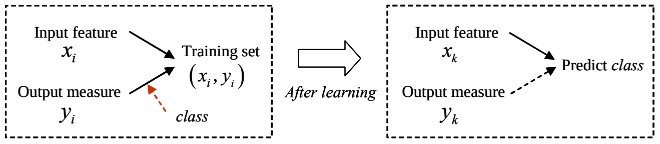

Figure 3. Concept of supervised learning.

The goal of the supervised learning is to learn a mapping from $x \in X$ to $y \in Y$, given training sets that consist of input and output pairs. Here $X = \{x_1, x_2, \ldots, x_n\}$ denotes a set of inputs (a.k.a. *input features*), and $Y = \{y_1, y_2, \ldots, y_n\}$ is a set of outputs (a.k.a. *output measures*). The input feature set contains quantifiable features of the system under consideration. The output measure set can be a continuous value (called *regression*) or a class label of the input (called *classification*), which thus results in a numerical or categorical measure. If the output measure is numerical (categorical), then the learning will become a regression (classification) problem.

In this paper, each output measure is labeled with a pre-defined class (e.g., performance level). The learning is performed on a collection of training sets. Thus, training an agent (e.g., a PM) involves finding a mapping from input features to output measures so as to enable the agent to accurately predict the class of an output measure when a new input feature is given. Figure 3 shows the concept of supervised learning, where the agent predicts the classes of output measures y_k when input features x_k are given after learning with the training sets, where $k = 1, \ldots, n$.

Considering algorithms for supervised learning, there are a number of methods for classification such as the rule learner, decision tree learner, Bayesian classifier, instance-based learner, and support vector machine [10]. In our problem setup, we have found that the Bayesian classifier is more efficient than other methods since it can efficiently classify the output features corresponding to a new input feature into a finite number of class labels. The key to speed of the classification step is the pre-computation of prior and conditional probabilities based on a training step (see below).

3.2 Learning-based Power Management Framework

It is useful to describe how the supervised learning can be adapted to the power management problem. Figure 4 presents a top level structure of the proposed PM which incorporates a Bayesian learning framework. The learning framework consists of two phases: training phase and classification phase. Essentially, we aim to use the supervised learning to enable the PM to automatically discover the relations between input features and output measures and to predict the processor's performance level (power dissipation and execution time per task) by using the classification.

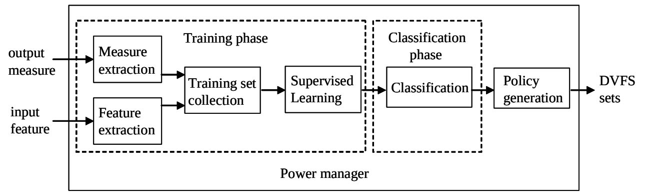

Figure 4. Structure of the proposed power manager.

Key functions implemented inside the PM are as follows:

- Feature extraction: choose the input feature (i.e., characteristics of the tasks, and state of the SQ),
- Measure extraction: choose the output measure (i.e., the power dissipation and execution time of the tasks),
- Training set generation: assemble the input feature and output measure into the training sets,
- Supervised learning: map the input feature to the output measure based on the training sets, and
- Classification: select the most likely class given the input feature.

The proposed supervised-learning DPM technique mainly comprises of three parts: training, classification, and policy generation. The procedures for training, classification and policy generation are explained next.

3.2.1 *Input Feature and Output Measure Extraction*

The first step is the training phase which extracts input features and output measures, where system knowledge is required to produce well-prepared training sets. During the process of feature extraction, in the context of the power management problem, the PM gathers input features such as the type of tasks (e.g., high-priority or low-priority), the state of the SQ, and the arrival rate of tasks, which affect the performance level of the SP. In addition, the PM collects performance-related information (e.g., the system power dissipation and the execution time of tasks) as the output measures. The class of each output measure, considered as an *attribute*, is as a pre-defined level or range, such as a range of system power dissipations or time durations for task execution.

Table 1 shows an example of training sets which consist of selected input feature and output measure pairs. Notice that the queue occupancy and the arrival rate of task are assigned attributes (i.e., low, med, or high), where low = [0 33%], med = (33% 67%], and high = (67% 100%] when applied to the SQ occupancy, and low = [0 0.33], med = (0.33 0.67], and high = (0.67 1] when applied to the arrival rate. Each output measure is labeled with a specific class from the set L. In our problem setup, the class set L is defined as $L_1 = \{pow_1, pow_2, pow_3\}$ where $pow_1 < pow_2 < pow_3$, and $L_2 = \{exe_1, exe_2, exe_3\}$ where $exe_1 < exe_2 < exe_3$. Note that each class is defined as a range of values, e.g., pow_1 = [34mW 41mW], pow_2 = (41mW 47mW], pow_3 = (47mW 54mW], exe_1 = [14.1ns 21.5ns], exe_2 = (21.5ns 28.5ns], and exe_3 = (28.5ns 35.7ns]. In addition to our input features, the power dissipation and execution time may be determined by many other factors, including the cache hit/miss ratio, cache hierarchy, and so on. However, the extent to which these factors impact the performance level of the SP is highly dependent on the system architecture or configuration, which is outside the scope of the present paper.

Table 1. Example training set for the DPM problem.

Input features			Output measures	
Task type	Queue occupancy	Arrival rate of task	Power dissipation	Execution time
high-priority	med	low	pow_2	exe_1
high-priority	low	med	pow_3	exe_1
low-priority	med	low	pow_2	exe_3
low-priority	low	med	pow_1	exe_3
high-priority	low	med	pow_1	exe_1
low-priority	med	med	pow_2	exe_2
low-priority	med	med	pow_1	exe_2
low-priority	med	high	pow_2	exe_2
low-priority	med	med	pow_1	exe_1

The training set size affects the accuracy of classification, i.e., variance of the predicted value increases as the training set size is reduced, resulting in an increased bias. In this paper, the training set size is determined by calculating a conditional probability while varying the set size, as described in the experimental results section.

3.2.2 *Training and Classification*

Having obtained the training set, the second step is the classification phase, which uses supervised learning to train an accurate classifier. The classifier's goal is to organize a new input feature $\{x_1, x_2, \ldots, x_n\}$ into a finite number of classes l from the set L for each one of the output features in the set $\{y_1, y_2, \ldots, y_n\}$.

Specifically, in the Bayesian classifier, the classification task is essentially the assignment of the *maximum a posteriori* (MAP) class given the data $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and the prior of class assignments to y_i by maximizing the *posterior probability* $Prob(y_i = l \mid x_1, x_2, \ldots, x_n)$ of assigning class l to output feature y_i given the new evidence $\boldsymbol{x}$, such as

$$\begin{aligned} y_{MAP} &= \arg\max_{l} Prob(y_i = l \mid x_1, x_2, \ldots, x_n) \\ &= \arg\max_{l} \frac{Prob(x_1, x_2, \ldots, x_n \mid y_i = l) \cdot Prob(y_i = l)}{Prob(x_1, x_2, \ldots, x_n)} \end{aligned} \quad (1)$$

The denominator $Prob(x_1, x_2, \ldots, x_n)$, which is the *marginal probability* of witnessing the new evidence $\boldsymbol{x}$ under all possible hypotheses, is irrelevant for decision making since it is the same for every class assignment. $Prob(y_i = l)$, which is the *prior* (pre-evidence) *probability* of the hypothesis that the class of y_i is l, is easily calculated from the training set. Hence, we only need $Prob(x_1, x_2, \ldots, x_n \mid y_i = l)$, which is the *conditional probability* of seeing the input feature vector $\boldsymbol{x}$ given that the class of y_i is l. The factor $\frac{Prob(x_1, x_2, \ldots, x_n \mid y_i = l)}{Prob(x_1, x_2, \ldots, x_n)}$ represents the impact of the new evidence $\boldsymbol{x}$ on the hypothesis that $y_i = l$. If it is likely that the evidence will be observed when this hypothesis is true, then this factor will be large. Multiplying the prior probability by this factor results in a large posterior probability of the hypothesis given the evidence. The Bayes' theorem thus measures how much new evidence should alter belief in some hypothesis.

Now, $Prob(x_1, x_2, \ldots, x_n \mid y_i = l)$ may be expanded as $Prob(x_1 \mid x_2, \ldots, x_n, y_i = l) \times Prob(x_2, x_3, \ldots, x_n \mid y_i = l)$. The second factor above can be decomposed in the same way, and so on. Furthermore, assuming that all input features are conditionally independent given the class, i.e., $Prob(x_1 \mid x_2, \ldots, x_n, y_i = l) = Prob(x_1 \mid y_i = l)$. Therefore, we obtain: $Prob(x_1, x_2, \ldots, x_n \mid y_i = l) = \prod Prob(x_j \mid y_i = l)$, and we compute the maximum a posteriori class as follows:

$$y_{MAP} = \arg\max_{l} Prob(y_i = l) \cdot \prod_{j=1}^{n} Prob(x_j \mid y_i = l) \tag{2}$$

An example of how to classify the input features is given next. Suppose that we have a set of three input features and a set of two output features as shown in Table 1, where $\{x_1, x_2, x_3\}$ = {task type, queue occupancy, arrival rate}, and $\{y_1, y_2\}$ = {power dissipation, execution time}. We first compute the per-input-feature conditional probabilities required for the classification task. For the example training set, we have: $Prob(x_1 = \text{low} \mid y_1 = pow_1) = Prob(x_1 = \text{low} \mid y_1 = pow_2) = 3/4$, $Prob(x_1 = \text{high} \mid y_1 = pow_1) = Prob(x_1 = \text{high} \mid y_1 = pow_2) = 1/4$, and $Prob(x_1 = \text{high} \mid y_1 = pow_3) = 1$. There may be some cases where particular input features do not occur together with an output measure due to an insufficient number of data points in the training set. In this case, a standard way to deal with zero conditional probabilities is to eliminate them by smoothing [13],

$$Prob(x_j \mid y_i = l) = \frac{freq(x_j, y_i = l) + \lambda}{freq(y_i = l) + \lambda n_x} \tag{3}$$

where λ is a smoothing constant ($\lambda > 0$), and n_x is the number of different attributes of x_i that have been observed. For the example training set, using equation (3) with $\lambda = 1$, we have: $Prob(x_1 = \text{low} \mid y_1 = pow_3) = Prob(x_2 = \text{med} \mid y_1 = pow_3) = 1/4$. We will also need the prior probabilities for the various output feature classifications, which are calculated from the training set data. In this example, $Prob(y_1 = pow_1) = Prob(y_1 = pow_2) = 4/9$, and $Prob(y_1 = pow_3) = 1/9$. After calculating the conditional and prior probabilities, the PM can decide the best power management policy by predicting the MAP class for a new input feature vector.

Let a new input feature (x_1 = low, x_2 = med, x_3 = med), which was not in the training set, be presented to the PM, which classifies the input feature based on equation (2) as follows. Firstly, for the hypothesis $y_1 = pow_1$, the posterior probability is: $Prob(y_1 = pow_1) \cdot Prob(x_1 = \text{low}, x_2 = \text{med}, x_3 = \text{med} \mid y_1 = pow_1) = (4/9)\cdot(3/4)\cdot(1/2)\cdot(1) = 1/6$ because $Prob(x_1 = \text{low} \mid y_1 = pow_1) = 3/4$, $Prob(x_2 = \text{med} \mid y_1 = pow_1) = 1/2$ and $Prob(x_3 = \text{med} \mid y_1 = pow_1) = 1$. Secondly, for the hypothesis $y_1 = pow_2$, the posterior probability is: $Prob(y_1 = pow_2) \cdot Prob(x_1 = \text{low}, x_2 = \text{med}, x_3 = \text{med} \mid y_1 = pow_2) = (4/9)\cdot(3/4)\cdot(1)\cdot(1/4) = 1/12$ because $Prob(x_1 = \text{low} \mid y_1 = pow_2) = 3/4$, $Prob(x_2 = \text{med} \mid y_1 = pow_2) = 1$ and $Prob(x_3 = \text{med} \mid y_1 = pow_2) = 1/4$. Lastly, for the hypothesis $y_1 = pow_3$, the posterior probability is: $Prob(y_1 = pow_3) \cdot Prob(x_1 = \text{low}, x_2 = \text{med}, x_3 = \text{med} \mid y_1 = pow_3) = (1/9)\cdot(1/4)\cdot(1/4)\cdot(1) = 1/144$ because $Prob(x_1 = \text{low} \mid y_1 = pow_3) = 1/4$, $Prob(x_2 = \text{med} \mid y_1 = pow_3) = 1/4$ and $Prob(x_3 = \text{med} \mid y_1 = pow_3) = 1$. Consequently, the MAP class of the power dissipation for the new input feature vector is pow_1. Similarly, computing MAP of the execution time results in posterior probabilities of hypotheses $y_2 = exe_1$, $y_2 = exe_2$, and $y_2 = exe_3$ being 1/24, 2/9, and 1/18. Thus, the PM concludes that the MAP class of the execution time is exe_2.

The PM predicts the MAP performance level of the processor when a new task arrives in the SQ. The classification based on the Bayesian classifier is robust to noisy and/or extraneous input features. It is also fast because it only requires a single pass through the training data to initialize the prior and conditional probabilities while requiring only a few multiplications and comparison to determine the MAP performance level of the processor at runtime.

3.2.3 Stochastic Policy Optimization

Finding an optimal power management policy in a learning-based framework requires an autonomous decision making strategy which maps the output classes to actions. The actions commanded by the PM change the performance state of the system and lead to quantifiable penalties (or rewards). In this paper, we consider the case where an action incurs a cost (e.g., energy dissipation), where the PM's goal is to devise a policy for issuing a command that minimizes this expected cost.

Assume that the target processor system has k (power-delay or PD for short) states denoted by $s_1, \ldots, s_k$, where $s_1 < \ldots < s_k$ in terms of the PD product (PDP) in the respective states. The PM can choose an action from a finite set of supply voltage-clock frequency (VF) settings $A = \{a_1, \ldots, a_n\}$, where $a_1 < \ldots < a_n$ in terms of the VF values (notice that a lower V requires a correspondingly lower F for the processor while a higher V allows a higher F, hence VF pairs may be considered as a single optimization variable in this setup).

There is a state transition probability for transitioning from state s to another state s' after executing an action a, i.e., $T(s', a, s) = Prob(s' \mid a, s)$. Furthermore, we make a common assumption that the cost function is additive (the PDP which is the same as energy dissipation is clearly additive). Considering the minimization of the total energy dissipation as an objective, we define the energy dissipation of a system at a given time t as follows. First, assume that the predicted classes for the output measures (i.e., power dissipation and execution time) are p and d, where $p \in L_1$ and $d \in L_2$ as defined in our problem setup. Note that p and d may be considered as ranges of power and execution time values, i.e., $p = [p_- \, p_+]$ and $d = [d_- \, d_+]$. Then, the expected cost of current state, $C(s, a)$, where a is the action prescribed by the PM in state s=<p, d>, is defined as a specific range such that

$$C(s,a) \in \left[p_- \cdot d_- + e(s,a) \quad p_+ \cdot d_+ + e(s,a) \right] \tag{4a}$$

where $e(s, a)$ is the expected energy dissipation to transit from state s to some next state under action a, which is in turn calculated from $T(s', a, s)$ and the state transition energy dissipation overhead. The above expression means that cost lies between expected minimum and maximum costs. To obtain a scalar cost function, we define:

$$C(s,a) = \frac{p_- \cdot d_- + p_+ \cdot d_+}{2} + e(s,a) \tag{4b}$$

We develop a policy generation technique by using well-known dynamic programming method making use of principles of overlapping subproblems, optimal substructures, and memoization. We speak of the minimum cost of a system state as the expected infinite discounted sum of cost that the system will accrue if it starts in that state and executes the optimal policy [14]. Generally, using π as a decision policy, this minimum cost is written as

$$\Psi^*(s) = \min_{\pi} E\left(\sum_{t=0}^{\infty} \gamma^t \cdot c(t) \right) \tag{5a}$$

where γ is a discount factor, $0 \le \gamma < 1$, and $c(t)$ is the cost at time t.

In our problem setup, the minimum cost function is unique and can be defined

$$\Psi^*(s) = \min_{a} \left(C(s,a) + \gamma \sum_{s' \in S} T(s', a, s) \Psi^*(s') \right) \quad \forall s \in S \tag{5b}$$

which asserts that the cost of a state s is the expected instantaneous cost plus the expected discounted cost of the next state, using the best available action. From Bellman's principle of optimality [15], given the optimal cost function, we specify the optimal policy as

$$\pi^*(s) = \arg\min_{a} \left(C(s,a) + \gamma \sum_{s' \in S} T(s', a, s) \Psi^*(s') \right) \tag{6}$$

Simply stated, the power manager determines the optimal action based on Eqn. (6) at each event occurrence (i.e., decision epochs). The task of casting the decision epochs to absolute time units is achieved by the system developer. Unlike AC line-powered systems, we focus on battery operated systems that strive to conserve energy to extend the battery life,.

Given $C(s, a)$ and $T(s', a, s)$, one way to find an optimal policy is to find the minimum cost function. It can be determined by an iterative algorithm (cf. Figure 5) called value iteration that can be shown to converge to the correct Ψ^* values. It is not obvious when to stop this algorithm. A key result bounds the performance of the current greedy policy as a function of the Bellman residual of the current cost function [16]. It states that if the maximum difference between two successive cost functions is less than ε, then the cost of the greedy policy (i.e., the policy obtained by choosing, in every state, the action that minimizes the estimated discounted cost, using the current estimate of the cost function) differs from the cost function of the optimal policy by no more than $2\varepsilon\gamma/(1-\gamma)$ at any state. This provides a stopping criterion for the algorithm.

```
1:  initialize $\Psi(s)$ arbitrarily
2:    loop until policy good enough
3:      loop for $\forall s \in S$
4:        loop for $\forall a \in A$
5:          $Q(s,a) = C(s,a) + \gamma \sum_{s' \in S} T(s',a,s)\Psi(s')$
6:          $\Psi(s) = \min_a Q(s,a)$
7:        end loop
8:      end loop
9:    end loop
```

Figure 5. The value iteration algorithm.

Results of the policy generation are stored in a state-action *mapping table* so that the PM does not need to compute the optimal action in each system state at runtime. Instead the optimal action generation is reduced to a simple table lookup. In practice, the PM examines the input features each time a new task arrives in the SQ, estimates the most likely state of the system, and looks up and issues the corresponding "optimal" action from the mapping table.

4. Experimental Results

We applied the proposed DPM technique to a multicore processor which includes a dynamic load balancing (DLB) block and four processors (cf. Figure 1). The DLB block, which guarantees in-order delivery of tasks, enables tasks from a single network interface to be processed in parallel on multiple processors. As a typical application, we performed TCP/IP-related tasks (e.g., TCP segmentation and checksum offloading [17]) on this processor.

In the first experiment, we analyzed performance characteristics of the processor in terms of the power dissipation and execution time. Considering the power number of the processor, we relied on detailed gate-level realization of the processor with TSMC 65nmLP library to precisely capture the power-saving opportunities. By varying the voltage and frequency values during the simulation setup, we achieved power and delay numbers with Power Compiler [18] for the processor after running the same tasks. We defined a set of three actions, i.e., a_1 = [1.00V, 150MHz], a_2 = [1.08V, 200MHz], and a_3 = [1.20V, 250MHz] for simplicity.

We generated a training set by running a set of tasks on the processor as follows. First, we considered a scenario whereby the processor accepts two types of tasks: low-priority and high-priority, where a high-priority task can move ahead of all the low-priority tasks waiting in the queue. Next, we defined a set of input features {occupancy state of the SQ, arrival rate of task} and output measures {power dissipation [mW], execution time [ns]}, similar to Table 1. The classes of output measures are defined as: pow_1, pow_2, and pow_3 = [34. 41.0], (41.0 47.0], and (47.0 54.0] as well as exe_1, exe_2, and exe_3 = [14.1 21.5], (21.5 28.5], and (28.5 35.7]. During the training phase, voltage and frequency values are assigned to the processor based on simple requirements such as:

- The processor runs faster when high-priority tasks arrive with medium or high arrival rates,
- The processor runs slower when low-priority tasks arrive with low or medium arrival rates.

Since the training set size affects the classification accuracy, we performed simulations to determine an appropriate size by varying the set size from 50 to 3000 as shown in Figure 6. We have thus empirically determined that a training set size of 1000 is adequate.

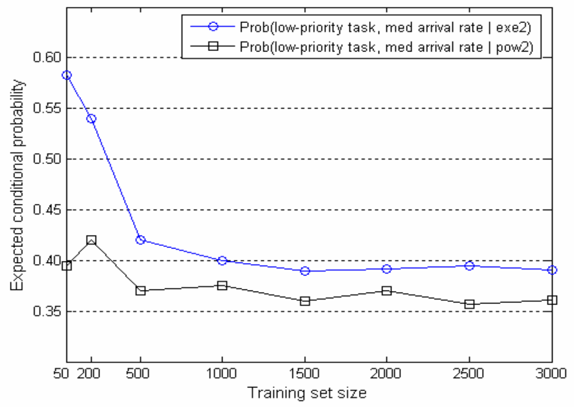

Figure 6. Selection of the training set size.

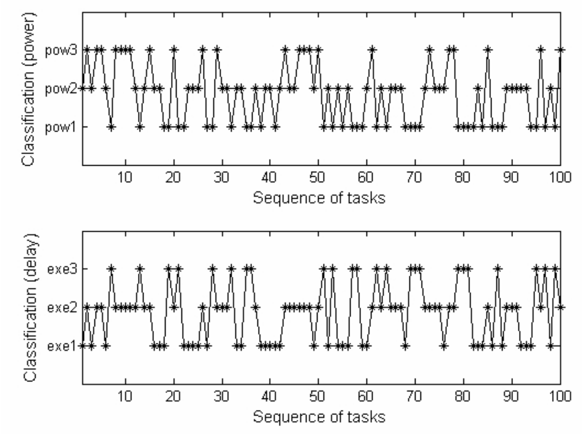

Figure 7. Classification for tasks in terms of power dissipation and execution time levels.

The second experiment was designed to demonstrate the effectiveness of the proposed learning-based DPM framework. The PM first collects the training sets which consist of 1000 input feature and output measure pairs. Second, based on the classes of the output measures, the PM calculates the required prior and conditional probabilities. Next, we randomly generated 100 tasks that arrive into the queue of the corresponding processor. Then, the classification was performed for each incoming task to determine the system state, i.e., power and time (i.e., delay) levels, as shown in Figure 7, where the x-axis represents the sequence of tasks (i.e., from task 1 to task 100). After predicting the system state, the PM looks up the pre-characterized system state to action mapping table to obtain the best action to issue.

Finally, we investigated the energy-efficiency of the proposed DPM technique. For comparison purpose, we implement a power management policy (denoted by Greedy), as described below. We use three VF values to simplify the experimental setup ($a_1 < a_2 < a_3$ in terms of VF values).

Greedy: Apply a greedy DPM assignment strategy which

- Uses the lowest a_1 value when $0 <$ arrival rate of task ≤ 0.33 (i.e., low workload).
- Likewise, it uses a_2 and a_3 when $0.33 <$ arrival rate of task ≤ 0.67 and $0.67 <$ arrival rate of task < 1, respectively.

Bayesian: Apply the optimal actions, based on the Bayesian learning-based DPM method described in this paper.

Next, we generated a number of tasks by selecting the priority and arrival rate of tasks randomly, where $0 <$ the arrival rate of tasks < 1, and applied the above-mentioned DPM policies to the multicore processor described earlier. The simulation results in Figure 8, which report the (normalized) energy dissipation of each task (numbered from 1 to 100) for one processor, show that the proposed DPM technique, i.e., **Bayesian** exhibits sizeable energy savings compared to the other techniques. Note that considering the performance of the processor, the overhead of performing classification in **Bayesian** is negligible since it does not affect the execution time of the processors (i.e., the classification and table lookup are performed during the queuing period before the VF change). Experimental results in Table 2, which also reports the characteristics of the workload distribution for each processor (e.g., the Proc1 has 27 high-priority tasks, etc.), demonstrate that, compared to the **Greedy** policy, the **Bayesian** policy achieves system-wide energy savings of up to 28.7% (these are the averages on four processors), respectively.

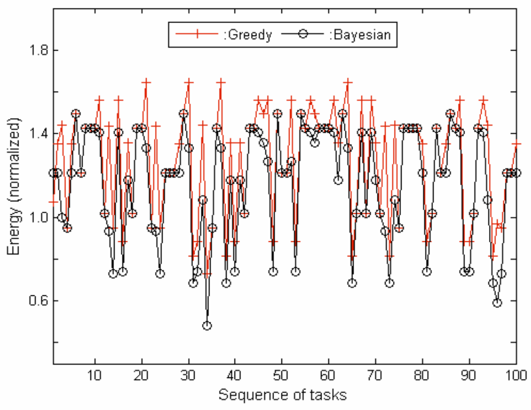

Figure 8. Energy dissipation comparison between a greedy DPM and the Bayesian Learning based DPM.

Table 2. Energy savings in the multicore processor.

Processor	Number of tasks			Total energy (normalized)		Energy saving Over Greedy
	High-pri	Low-pri	Total	Greedy	Bayesian	
Proc1	27	23	50	77.8	64.1	17.6%
Proc2	52	48	100	170.9	113.8	33.4%
Proc3	67	83	150	258.4	175.7	32.1%
Proc4	103	97	200	340.9	231.5	32.0%

5. Conclusions

This paper addressed the problem of dynamic power management, where a system-level PM continually intervenes to exploit power-saving opportunities subject to performance requirements. The overhead associated with regular activity of the PM to monitor the workload of a system and make decisions about power management of different functional blocks in the system tends to undermine the overall power savings of the DPM approaches. This paper thus described a supervised learning based DPM framework for a multicore processor, which enables the PM to predict the performance state of the system for each incoming task by a simple and efficient analysis of some readily available input features. Results demonstrate the effectiveness of the framework.

References

[1] Y-H. Lu, and G. De. Micheli, "Comparing System-Level Power Management Policies," *IEEE Design & Test of Computers*, Vol. 18, Issue 2, Mar-Apr. 2001.

[2] A. Naveh, et al., "Power and Thermal Management in Intel Core Duo Processor," *Intel Technology Journal*, Vol. 10, Issue 2, May, 2006.

[3] Dual-Core Processor Power and Thermal Design Guide, Mar., 2006. http://www.amd.com

[4] A. Iyer, and D. Marculescu, "Power Efficiency of Voltage Scaling in Multiple Clock, Multiple Voltage Cores," *Proc. of Int'l Conf. on Computer Aided Design*, Nov., 2002.

[5] R. Kumar, et al., "Single ISA Heterogeneous Multicore Architecture: The Potential for Processor Power Reduction," *Proc. of 36th Symposium on Microarchitecture*, Dec., 2003.

[6] Q. Wu, P. Juang, M. Martonosi, and D.W. Clark, "Voltage and Frequency Control with Adaptive Reaction Time in Multiple-Clock Domain Processors," *Proc. of 11th Symposium on HPCA*, Feb., 2005.

[7] K. Lee, et al, "Low-power Network-on-Chip for High-Performance SoC design," *IEEE Trans. on VLSI*, Vol. 14, Issue 2, Feb., 2006.

[8] E. Chung, L. Benini, and G. De Micheli, "Dynamic Power Management Using Adaptive Learning Tree," *Proc. of ICCAD*, Nov., 1999.

[9] G. Dhiman, and T. S. Rosing, "Dynamic Power Management Using Machine Learning," *Proc. of ICCAD*, Nov., 2006.

[10] O. Chapelle, B. Scholkopf, and A. Zien, *Semi-Supervised Learning*, The MIT Press, 2006.

[11] J. Silc, B. Robic, and T. Ungerer, *Processor Architecture: From Dataflow to Superscalar and Beyond*, Springer, 1999.

[12] T.D. Burd, and R.W. Brodersen, "Design Issues for Dynamic Voltage Scaling," *Proc. of ISLPED*, Aug., 2000.

[13] Tom Mitchell, *Machine Learning*, McGraw Hill, 1997.

[14] A. Gosavi, *Simulation-based Optimization: Parameter Optimization Techniques and Reinforcement Learning*, Kluwer Academic, 2003.

[15] R.E. Bellman, *Dynamic Programming*, Princeton University Press, 1957.

[16] R J. Williams, and L C. Baird, "Tight performance bounds on greedy policies based on imperfect value functions," Technical Report NU-CCS-93-14, Northeastern University, Nov., 1993.

[17] IEEE 802.3 Ethernet document. http://www.ieee802.org.

[18] Synopsys compiler. http://www.synopsys.com.