Supervised Learning Based Power Management for Multicore Processors

Hwisung Jung, *Student member* and Massoud Pedram, *Fellow*, *IEEE*

Abstract - **This paper presents a supervised learning based power management framework for a multi-processor system, where a power manager (PM) learns to predict the system performance state from some readily available input features (such as the occupancy state of a global service queue) and then uses this predicted state to look up the optimal power management action (e.g., voltage-frequency setting) from a precomputed policy table. The motivation for utilizing supervised learning in the form of a Bayesian classifier is to reduce the overhead of the PM which has to repetitively determine and assign voltage-frequency settings for each processor core in the system. Experimental results demonstrate that the proposed supervised learning based power management technique ensures system-wide energy savings under rapidly and widely varying workloads.**

Index Terms — **Bayesian classification, dynamic power management, machine learning, multi-processor system, supervised learning**

I. INTRODUCTION

Ongoing demand for high performance – yet thermally sustainable - processing have resulted in the introduction of chip multiprocessor architectures to enable continued performance scaling without having to increase the chip clock frequencies beyond a few GHz. At the same time, there are strong motivations (i.e., dollar cost of energy consumption, thermal power budget constraint, service life of the system in between batter recharges in case of mobile platforms) to make multi-core processing platforms power and energy efficient.

Conventional dynamic power management (DPM) methods have not been able to take full advantage of power-saving techniques such as dynamic voltage and frequency scaling (DVFS). This is because i) the system-level power manager has a limited opportunity to utilize DVFS due to the energy and delay overheads incurred during power mode transitions [1], and ii) the power management algorithm (process), which continuously monitors the workloads of multiple processors, analyzes the information to make decisions, and issues DVFS commands to each processor, can give rise to a considerable computational overhead and/or complicate the task scheduling [2]. The higher the number of cores in the processor is, the more severe these issues become. Therefore, the ability of a DPM framework to scale well on a multicore processor by eliminating these overheads is becoming a critical requirement [3][4].

In the literature, DPM is typically referred to a strategy whereby a resource manager (hardware, firmware, or the operating system) turns of or off the processing cores when they are idle (or new tasks arrive). In contrast, DVFS is defined as a technique which dynamically varies the supply voltage and operating frequency values applied to the processing cores in response to load conditions or workload characteristics. It is easy to see that DPM can be easily combined with DVFS, i.e., a power manager may not only issue commands to various processing cores to turn on or off, but also change their power-performance state by issuing DVFS commands. In our paper, however, we do not consider power gating as an option, i.e., when we speak of DPM, we mean DPM using DVFS as the power optimization level only.

The problem of determining a power management policy that applies DVFS to a multicore processor has recently received a lot of attention – see, for example, [5]-[10]. Although these techniques perform system-level DPM or DVFS for multicore processors, little attention has been paid to improve decision-making strategy which minimizes the 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 (i.e., quantifiable features of the system under consideration) and accurately predict the overall system performance state, which is subsequently used to choose and issue the "optimal action".

Traditional approaches for DPM, which are based on models of service requestor (SR), service provider (SP), and service queue (SQ), tend to work very well if the workload of the system does not change rapidly. In such a case, the energy and delay overheads of power mode transitions can become quite significant, rendering the DPM strategy ineffective. Indeed, adaptive power management techniques are unsuccessful in reducing the total chip power dissipation when the overhead of power-mode transitions is not controlled in a multicore processor, which is subjected to frequent changes in the load conditions [10]. Our thesis is that knowing (or predicting) in real time which frequency and voltage levels to use, and when to apply a new performance setting in a multicore processor, must be done with the aid of a self-improving (i.e., intelligent and autonomous) power manager that can detect the load conditions and react appropriately.

In this paper, we address a dynamic power management problem where a PM continuously issues power mode transition commands to maximally exploit the power-saving opportunities. The overhead associated with the functioning of the PM to monitor the workload of the system and make decisions about performance mode (voltage and frequency level) of different cores in a multicore processing system tends to be high. This paper thus describes a supervised learning [11] based DVFS for the multicore processor, which enables the PM to predict the performance state of the processor for each incoming task by inspecting some readily available input features, followed by a Bayesian classification technique.

Supervised learning (SL) refers to the formal theory of developing computational models for learning behaviors of agents as part of the machine learning discipline [11][12]. The key rational for utilizing SL for power management is to reduce the overhead of the PM. Experimental results demonstrate the effectiveness of the proposed power management framework and show that it achieves sizeable system-wide energy savings under rapidly varying workloads in a wired communication application scenario.

In the remainder of this paper we use the terms chip multi processor (CMP) and multi-processor system (MPS) interchangeably. Moreover, *we assume that the different cores within a CMP or the different processors within a MPS can be independently turned on/off or voltage and frequency scaled.* We realize that the current generation of CMP designs (see for example Intel Nehalem [53]) do not offer per-core dynamic voltage and frequency scaling, but expect that the future generations of the CMP designs will support this important power/performance scaling feature. Regardless, the proposed approach can be applied to different processors in a MPS (e.g., a Blade server used in datacenters).

The remainder of this paper is organized as follows. Section II provides background of this paper while section III describes some related work. Section IV provides the details of proposed supervised learning based power management framework. An extraction strategy for input features and output measures is described in section V. In section VI, we present a stochastic policy optimization technique. Experimental results and conclusion are given in section VII and section VIII.

II. Background

Consider a power-managed MPS, where each processor is equipped with multiple power-saving modes (i.e., different DVFS settings). A system-level PM dynamically assigns the DVFS setting for each processor based on its workload as is shown in Fig. 1 for a distributed shared-memory MPS.

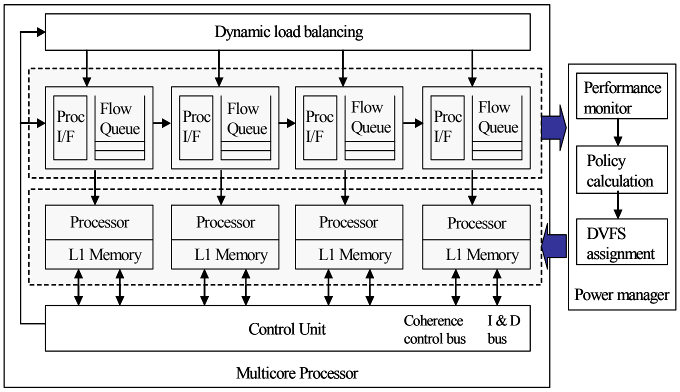


Fig. 1. Example of a power-managed multi- processor system.

The figure also shows a *dynamic load balancing block* which enables high-throughput and low-latency data flow for each processor and a *control unit* which ensures cache coherency. The flow queue (i.e., receive queue) interacts with the PM by providing information about a processor's workload for the purpose of controlling the performance state of the processor. The PM, which profiles and analyzes the workload characteristics i.e., the arrival rate of tasks by examining the flow queue, determines and executes a power management policy (i.e., one that maps workloads to power state transition commands) so as to minimize the system energy dissipation. Details of the processor functionality are omitted here for brevity. Interested readers may refer to [13][14][15].

When tasks are given to a MPS, the dynamic load balancing block (i.e., SR) dispatches each task into some flow queue (i.e., local SQ). Each processor (i.e., SP) reads the assigned tasks from its SQ. At regular time instances (or aperiodic times dictated by interrupts), called *decision epochs*, the PM determines the workload of the processor by checking the occupancy state of its SQ, and subsequently, assigns a DVFS value to the processor. Note that the decision epochs are separated by a fixed (or some average) time interval; the shorter this time interval is, the higher the delay and energy dissipation overheads of the PM are. This is because the DVFS method utilizing a DC-DC converter with multiple regulated output voltage levels and a PLL with multiple output frequencies incur non-negligible mode transition latency and energy overheads [16]. At the same time, the shorter this interval is, the more responsive the PM is to changes in the workload. The shortcoming of the conventional DVFS procedure is the following. When the workload (the occupancy number of the SQ) changes, each processor has to send an interrupt to the PM to request a DVFS adjustment for the corresponding processor, which significantly increases the computational overhead of the PM in a MPS with a large number of processors. Alternatively, the PM on a regular basis examines the state of the SQ in front of each processor in order to determine the DVFS value for that processor, and subsequently, schedules a sequence of DVFS assignments for every processor. Either approach creates a significant overhead. A key contribution of our work is that an incoming task is directly labeled with an optimal DVFS value through the Bayesian classification process while it is still in the SQ.

III. Related Work

Dynamic power management techniques based on machine leaning [18] have been the subject of a number of recent investigations [21]-[25]. In the following, we provide a quick review of some works that are directly related to ours.

An adaptive power management technique based on machine learning was presented in [21], where the authors described a system that learns when to turn off functional blocks of the system based on different usage patterns, e.g., history of active application or the CPU utilization factor. In this model-based approach, system dynamics and user patterns are captured to choose power-saving actions.

The authors in [22][23] described a power management technique that employs a machine learning algorithm to choose an optimal policy from a set of power management policies available to a system. The proposed algorithm, which relies on processor runtime statistics, evaluates performance of the policies during each idle period to decide which policy to adopt next. Our proposed technique differs from [23] in that we use a supervised learning algorithm for deriving a self-improving policy.

An automated approach to identify a task-specific power management policy was proposed in [24], where an enforcement-learning based operating system automatically learns which action to take for a specific workload given to a system. The authors applied the proposed technique to hard disk power management in a mobile device, enabling the operating system to record hard disk accesses and monitor I/O related system parameters. In this approach, a classification algorithm that dynamically selects an appropriate spin-down policy is implemented.

The authors of [25] presented a machine learning approach to perform dynamic voltage scaling (DVS) on an integrated CPU-core and on-chip L2-cache. The proposed approach identifies application phases at runtime and issues appropriate DVS commands. The DVS policy itself is derived through a learning process performed on a representative workload. More precisely, first a training data set is generated by representing the workload as a CPU/cache frequency combination and the optimal DVS command for each such combination. Next a machine learning technique is applied to obtain a policy in the form of propositional (if-then) rules.

All of the above-mentioned power management approaches are based on machine learning techniques, where an agent (i.e., power manager) is trained based on a number of representative workloads or user patterns in order to learn the performance state of a target system for the purpose of taking a DVS or DVFS action. Unfortunately, little attention has been paid to power management policy optimization under a cost function and to the accurate classification of the performance state of the system. Furthermore, as explained previously, the aforesaid techniques are inefficient for MPS architecture due to computational overheads for deriving an optimal policy for each processor, exacerbating with scheduling of a series of DVFS assignments for every processor.

IV. Learning-based Power Management Framework

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

A. Background on Supervised Learning

Supervised learning [11] 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.

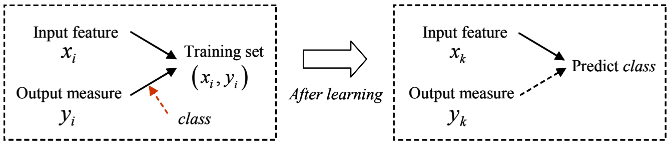


Fig. 2. Concept of supervised learning.

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 input features, and $Y = \{y_1, y_2, \ldots, y_n\}$ is a set of outputs 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. Fig. 2 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 $k = 1, \ldots, n$.

The key steps of the supervised learning may be stated as follows:

i) Determine inputs and outputs of the learner: Relevant input features and output measures (and the corresponding class labels) are chosen,
ii) Generate the training set: The training set – which is simply a collection of input features and corresponding output features and class labels - is designed so as to capture the important characteristics of the system,
iii) Training: This step results in the design of the classifier based on the training set,
iv) Classification: The classifier is used on arbitrary input features to predict the class labels of the output measures.

Considering algorithms for supervised learning, there are a number of methods for classification such as rule based learner, decision tree based learner, instance based learner, probability based learner, and kernel based learner. Details of each classification learner are omitted here for brevity. Interested readers may refer to [26][27][28][29].

In our problem setup, we have found that the probability based learner (i.e., 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).

B. Learning-based Power Management Framework

It is useful to describe how the supervised learning can be adapted to the power management technique. Fig. 3 presents a top level structure of the proposed PM which incorporates a Bayesian learning framework. The learning framework consists of two phases: extraction and classification phases.

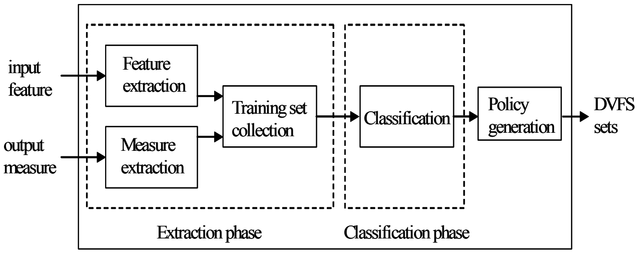


Fig. 3. Structure of the proposed power manager.

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. Key functions implemented inside the PM are as follows:

- Feature extraction: choose the input feature (i.e., characteristics of the tasks and the 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 feather 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-based power management technique mainly comprise of three parts: extraction, classification, and policy generation. The procedures for extraction and classification are explained next. Details of the extraction strategy for input features and output measures are further described in section IV, whereas the policy optimization technique is presented in section V.

1) Input Feature and Output Measure Extraction

The first step is the extraction 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 observes 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 I
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

TABLE I 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. The extent to which these factors impact the performance of the SP is highly dependent on the architecture and/or the system configuration (e.g., whether the SP's allow single or multiple thread execution). In this paper, we consider single-threaded core architectures only.

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.

2) Classification

The goal of classification is to predict the most likely class label of the output features given the input features. In the context of PM for a CMP system, the goal is to devise a power management policy for issuing DVFS commands that minimize the total energy dissipation of the CMP system based on the load conditions and workload characteristics.

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

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

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. Note that 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: $Prob(x_1, x_2, \ldots, x_n \mid y_i = l) = \prod_j Prob(x_j \mid y_i = l)$, and we compute the maximum a posteriori class as follows:

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

When used in real applications, the Bayesian classifier first partitions the training set into several subdatasets by the class label of the target output measure. Then, in each subdataset labeled by l for output measure y_i, the maximum likelihood (ML) estimator $Prob(x_j = a_{jk} \mid y_i$=$l)$ can be given by the frequency $\mathrm{n_{jkl}} / \mathrm{n_l}$, where n_{jkl} is the number of the occurrences of the event $\{x_j = a_{jk}\}$ in subdataset denoted by class label l; n_l is the number of the samples in the same subdataset.

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 [18] as follows

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

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.

i) 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 $Prob(x_3 = \text{med} \mid y_1 = pow_1) = 1$.
ii) 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 $Prob(x_3 = \text{med} \mid y_1 = pow_2) = 1/4$.
iii) 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) Discriminative Bayesian Classifier

As we have seen above, a Bayesian classifier assumes a conditional independency among the input features. When used for classification, the Bayesian classifier predicts a new data point as the class with the highest posterior probability by writing the classification rule in a decomposable form using the conditional independence assumption (see equation (2)).

A key advantage of the Bayesian classifier is the ability to deal with the missing information during classification (i.e., missing input features that are relevant to the identification of output features). For example, some information such as cache miss statistics or branch mis-prediction rate, which affect the processor performance are considered as missing input features in our problem setup. Assume the input feature set $\{x_1, x_2, \ldots, x_n\}$ be X. When the values of a subset of X, for example M, are unknown or missing, the marginalization inference can be obtained immediately as follows:

$$y_{MAP} = \arg\max_{l} Prob(y_i = l) \cdot \prod_{j \in X - M} Prob(x_j \mid y_i = l) \qquad (4)$$

No further computation is needed in handling this missing information problem, because each term $Prob(x_j \mid y_i$=$l)$ has been calculated in training the Bayesian classifier. However, there are shortcomings in this simple classifier. More precisely, this approach models the joint probability in each subset separately and then applies the Bayes rule to obtain the posterior classification rule. Consequently, this construction procedure - sometime called a *generative classifier* - discards some discriminative information for classification [17]. Without considering the other classes of data, this method only tries to approximate the information within each subdataset. On the other hand, a *discriminative classifier*, which directly estimates $Prob(y_i|x_j)$, preserves inter-subdataset information well by directly constructing decision rules among all available data [18]. Therefore, the Bayesian classifier may be extended to provide a global scheme to preserve the discriminative information among all the data. See [19] for a detailed description of a discriminative Bayesian classifier, which combines both merits of discriminative methods (e.g., support vector machines [20]) and the simple Bayesian classifiers described above. A more detailed discussion of discriminative Bayesian classifiers falls outside the scope of the present paper.

V. Extraction Strategy

In this section, we present the extraction strategy for input features and output measures.

A. Extracting Input Features

Input feature selection plays an important role in the classification procedure which maps input features onto output measures. There are some relevant input features that have important information regarding the output measures, whereas there may be some irrelevant ones containing little information regarding the output measures. Finding every input feature that contains relevant information about the resulting output measure is difficult and in many cases unnecessary. For example, capturing the amount of cache interference experienced among tasks that are co-scheduled on the same shared cache is difficult. Typically, a task is written to expose software “threads” of execution; the OS then maps these threads onto processors in the case of MPSs. The PM gathers available information on input features (e.g., types of the tasks, state of the SQ, and arrival rate of tasks) as explained in the previous section. At the same time, the PM needs to watch for the *missing input features* (e.g., the amount of cache interference) which affect the performance-related output measures as well.

There are two approaches to compensate for the missing input features [32]: *input feature-compensate method* and *classification-compensate method*. The first approach estimates values of hidden input features by using the expectation-maximization (EM) algorithm [33] and then performs classification on the complete input features. Note that the EM algorithm is a general technique that can be used to determine the maximum likelihood estimate (MLE) of the parameters of an underlying distribution from some given data when the measured data is incomplete. The second approach passes the incomplete input features directly to the classifier which is then adjusted to operate on the incomplete input features. A brief description of each method follows.

1) Input Feature-Compensate Method

Let x denote the known (measured) input feature and let m denote the missing input feature. Together x and m form the complete input feature. Notice that m can be a hidden source of variation that affects the output measures. Then, we have $Prob(x, m \mid \theta)$, the joint probability density function of the complete input features with parameters given by vector θ (θ may for example correspond to the mean value and variance of a Gaussian distribution). This function can also be considered as the complete data likelihood, that is, it can be thought of as a function of θ and expressed as

$$Prob(x, m \mid \theta) = Prob(m \mid x, \theta) \cdot Prob(x \mid \theta) \qquad (5)$$

by using the Bayes rule.

The EM algorithm iteratively improves an initial estimate θ^0 by constructing new estimate θ^1, θ^2, etc., where an individual re-estimation step that derives θ^{n+1} from θ^n takes the following form:

$$\theta^{n+1} = \arg\max_{\theta} Q(\theta) \qquad (6)$$

where $Q(\theta)$ is the expected value of the log-likelihood of complete input feature. Since we do not know the complete data, we cannot determine the exact value of the likelihood, but given the input feature x, we can calculate a posteriori estimates of the probabilities for the various values of m. For each set of m values, there is a likelihood value for θ, and we can hence calculate an expected value of the likelihood with the given values of x’s. Q is given by

$$Q(\theta) = \underset{m}{E}\left(\log Prob(x, m \mid \theta) \middle| x\right) \qquad (7)$$

where it is understood that this denotes the conditional expectation of $\log Prob(x, m \mid \theta)$ being taken with the θ used in $Prob(m \mid x, \theta)$ fixed at θ^n. In other words, θ^{n+1} is the value that maximizes the conditional expectation of log-likelihood of the complete input feature given the measured variables under the previous parameter values. The expectation $Q(\theta)$ may be rewritten as:

$$Q(\theta) = \int_{-\infty}^{\infty} Prob(m \mid x) \log Prob(x, m \mid \theta) dm \qquad (8)$$

These two steps (Expectation and Maximization) are repeated until $|\ \theta^{n+1} - \theta^n| \leq \omega$, where ω is some user specified tolerance level [34]. It can be shown that the EM iteration does not decrease the measured input feature likelihood function. The EM algorithm finds θ that maximizes the complete-input feature likelihood, which in turn removes the effect of hidden variables (i.e., the missing input features).

2) Classification-Compensate Method

In this method, the incomplete input features are used directly for the classification. Every input feature x is assigned a probability α to show how reliable and critical it is for the output measure. Likewise, each of the missing input features is assigned a probability $(1 - \alpha)$. Assuming that all measured input features and missing input features are independent, the total likelihood of each input feature simply becomes a weighted sum of the likelihood of the input features. Mathematically, this can be expressed as

$$
\begin{aligned}
&Prob(x_1, x_2, \ldots, x_n \mid y_i = l) \\
&= \prod_{j=1}^{n} \left(\alpha \cdot Prob(x_j \mid y_i = l) + (1-\alpha) \cdot Prob(m_j \mid y_i = l) \right)
\end{aligned} \tag{9}
$$

where y is the output measure and l is the class, provided that we have the missing input features $\boldsymbol{m} = (m_1, m_2, \ldots, m_n)$. In practice, we substitute (9) into (2) to compute the maximum a posteriori (MAP) during the classification.

B. Extracting Output Measures

Modern processors include hardware features for monitoring performance characteristics of the processor [30], which enables the PM to collect performance-related information. When an application runs by itself on a single processor system, the resources in that system are dedicated to its execution. Thus it is relatively easy to truthfully characterize and model resultant application performance behavior. However, when multiple applications run simultaneously on a MPS, it is comparatively difficult to determine the resources that end up being given to each individual application, which means that the performance behavior of each application on the MPS may not be measured accurately. Thus, the PM is forced to *observe* the output measure in a probabilistic way.

Let r denote an input feature state (r_i, i=1,…, h) where state r corresponds to a particular assignment of various attributes to input features ($x_1, x_2, \ldots, x_n$). Let o denote an observation which corresponds to output measures ($y_1, y_2, \ldots, y_n$) with various classes. Fig. 4 (a) illustrates observations for each output measure given an input feature state. Note that $o_{y1}(r_1)$ represents the observation o in y_1 (output measure) given the input feature state r_1. For example, the power dissipation (o_{y1}), one of output measures under consideration, of a processor given an input feature state r_1 (e.g., low priority task, medium queue occupancy, and high arrival rate of task) is normally distributed with mean of 38mW and variance of 2 i.e., N(38, 2).

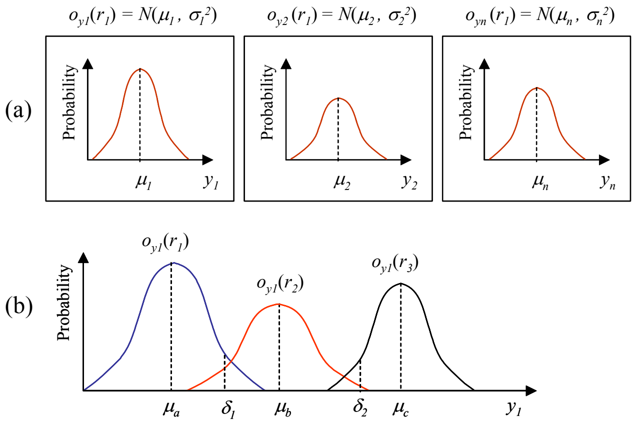


Fig. 4. (a) Observations for each output measure, and (b) Decision boundaries for an output measure among various input feature states

For accurate classification, the decision boundaries of the output measure in Bayesian classifier have to coincide with or be close to the performance specification criteria or boundaries. Fig. 4 (b) shows an example of decision boundaries for an output measure (e.g., o_{y1}) among various input feature states (e.g., r_1, r_2, and r_3), where our goal here is to find *the distinction points* δ_1 and δ_2.

By doing so, we can define the class as a range of values, as explained before. Let fr_1, fr_2, and fr_3 denote the probability density functions of output measure for the input feature states r_1, r_2, and r_3, respectively. Based on the illustration (see Fig. 4 (b)), δ_1 and δ_2 are determined from the following:

$$
\int_{-\infty}^{\delta_1} fr_1(x)dx = \int_{\delta_1}^{\infty} fr_2(x)dx \tag{10a}
$$

$$
\int_{-\infty}^{\delta 2} fr_2(x)dx = \int_{\delta_2}^{\infty} fr_3(x)dx \tag{10b}
$$

Assuming normal distribution function for the output measure in our problem setup, we can rewrite (10a) and (10b) as:

$$
\frac{1}{\sqrt{2\pi\sigma_a}} \int_{-\infty}^{\delta_1} e^{-\frac{(x-\mu_a)^2}{2\sigma_a^2}} dx = \frac{1}{\sqrt{2\pi\sigma_b}} \int_{\delta_1}^{\infty} e^{-\frac{(x-\mu_b)^2}{2\sigma_b^2}} dx \tag{11a}
$$

$$
\frac{1}{\sqrt{2\pi\sigma_b}} \int_{-\infty}^{\delta_2} e^{-\frac{(x-\mu_b)^2}{2\sigma_b^2}} dx = \frac{1}{\sqrt{2\pi\sigma_c}} \int_{\delta_2}^{\infty} e^{-\frac{(x-\mu_c)^2}{2\sigma_c^2}} dx \tag{11b}
$$

where μ_a, μ_b, and μ_c are the mean values of the output measure for the input feature states, and σ_a, σ_b, and σ_c are their standard deviations. Solving these integral equations, we obtain:

$$
\delta_1 = \frac{\mu_a\sigma_b + \mu_b\sigma_a}{\sigma_a + \sigma_b}, \quad \delta_2 = \frac{\mu_b\sigma_c + \mu_c\sigma_b}{\sigma_b + \sigma_c} \tag{12}
$$

TABLE II shows an example of the decision boundaries for various probability density functions of the output measure (i.e., power dissipation), while varying values of standard deviations, where $o_{y1}(r_1) = N(\mu_a, \sigma_a^2)$, $o_{y1}(r_2) = N(\mu_b, \sigma_b^2)$, and $o_{y1}(r_3) = N(\mu_c, \sigma_c^2)$, where each case is represented graphically in Fig. 5. To simplify the comparison among these, we assume that the mean values for the output measure are fixed (e.g., μ_a =37.5, μ_b = 44.0, μ_c = 50.5).

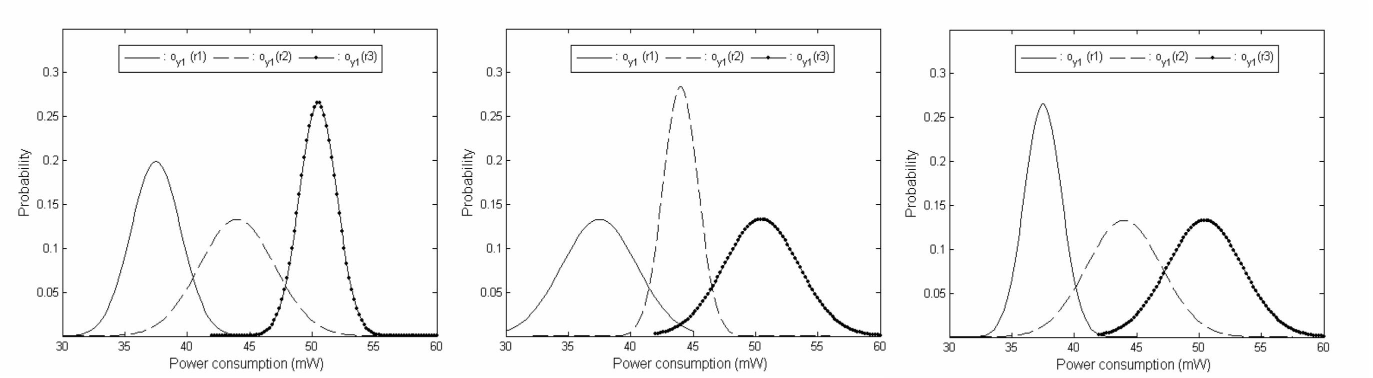
TABLE II
EXAMPLES OF DECISION BOUNDARIES

	case (a)			case (b)			case (c)		
	σ_a	σ_b	σ_c	σ_a	σ_b	σ_c	σ_a	σ_b	σ_c
	2.0	3.0	1.5	3.0	1.4	3.0	1.5	3.0	3.0
δ_1	40.1			41.9			39.6		
δ_2	48.3			46.0			47.3		
DI_1	1.30			1.47			1.44		
DI_2	1.44			1.47			1.08		

Without loss of generality, we assume, μ_b > μ_a. Next we introduce “distinction index (DI)” [35] as the performance criterion for boundary selection in output measure by the following:

$$
DI = \frac{\mu_b - \mu_a}{\sigma_b + \sigma_a} \tag{13}
$$

which indicates that the larger the value of DI is, the better the distinction between the output measures will be. For example, in case (c), DI_1 that represents the distinction between $o_{y1}(r_1)$ and $o_{y1}(r_2)$ is 1.44, which is greater than DI_2 (between $o_{y1}(r_2)$ and



(a) (b) (c)

Fig. 4. Examples of decision boundaries for various probability density functions of output measures (cf. TABLE II).

$o_{y1}(r_3)$). This indicates that we can achieve better accuracy in classification when we are given input feature states r_1 and r_2 rather than r_2 and r_3.

In conclusion, to ensure high accuracy in classification, the selection of distinction points has to be considered for the establishment of the discriminant function of the classifier.

VI. Power Management Policy

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). 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

$$C(s,a) \in \left[p_- \cdot d_- + e(s,a) \quad p_+ \cdot d_+ + e(s,a) \right] \tag{14a}$$

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:

$$C(s,a) = \frac{p_- \cdot d_- + p_+ \cdot d_+}{2} + e(s,a) \tag{14b}$$

We develop a policy generation technique by using well-known dynamic programming method making use of principles of overlapping subproblems, optimal substructures, and memorization. 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 [36]. Generally, using π as a decision policy, this minimum cost is written as

$$\Psi^*(s) = \min_{\pi} E\left(\sum_{t=0}^{\infty} \gamma^t \cdot c(t) \right) \tag{15a}$$

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

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

$$\Psi^*(s) = \min_a \left(C(s,a) + \gamma \sum_{s' \in S} T(s',a,s) \Psi^*(s') \right) \quad \forall s \in S \tag{15b}$$

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 [37], given the optimal cost function, we specify the optimal policy as

$$\pi^*(s) = \arg\min_a \left(C(s,a) + \gamma \sum_{s' \in S} T(s',a,s) \Psi^*(s') \right) \tag{16}$$

Simply stated, the power manager determines the optimal action based on equation (16) 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.

```
initialize $\Psi(s)$ arbitrarily
  loop until a stopping criterion is met
    loop for $\forall s \in S$
      loop for $\forall a \in A$
        $Q(s,a) = C(s,a) + \gamma \sum_{s' \in S} T(s',a,s)\Psi(s')$
        $\Psi(s) = \min_{a} Q(s,a)$
      end loop
    end loop
  end loop
```

Fig. 5. The value iteration algorithm.

Given $C(s, a)$ and $T(s', a, s)$, another way to find an optimal policy is to find the minimum cost function. It can be determined by an iterative algorithm (cf. Fig. 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 [38]. 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.

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.

VII. Experimental Results

A. Experimental Setup

We apply the proposed DPM technique to a multicore network processor which includes a dynamic load balancing (DLB, a.k.a., Application Delivery Controller or ADC) block and four processing cores (cf. Fig. 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 cores. There are various ways to distribute incoming tasks (a.k.a. connections or requests) to cores (a.k.a. back-end service hosts or servers), including the following methods [39]:

- *Least workload*: assigns the task to the host with the least workload (connections),
- *Fastest host*: assigns the task to the core that currently has the best performance,
- *Observed performance*: assigns the task to a core that has the highest performance rating, based on a combination of least workload and best response time,
- *Predictive method*: assigns the task to a core that has the highest predicted performance rating over time, and
- *Dynamic ratio*: determines the capabilities of the core to create a dynamic performance ratio accounting for host affinity to a connection and the resultant cache locality; the tasks are then distributed to the cores based on this ratio.

Among these, we consider RSS (receiver-side scaling) [40], which falls in the category of dynamic ratio techniques.[1] The RSS technique is capable of re-balancing the received processing load across multiple processor cores while maintaining in-order delivery of the data. RSS enables in-order packet delivery by ensuring that packets for a single connection are always processed by one processor. This RSS feature requires that the network adapter examine each packet header and then use a hashing function to compute a signature for the packet. To ensure that the load is balanced across the cores, the hash result is used as an index into an indirection table. Because the indirection table contains the specific core that is to run the associated deferred procedure call and the host protocol stack can change the contents of the indirection table at any time, the host protocol stack can dynamically balance the processing load on each core. As a typical application, we execute TCP/IP-related tasks (e.g., TCP segmentation and checksum offloading [41]). We vary the workload by changing the packet size from 64 bytes (e.g., 338,000 packets/sec) to 1,025 bytes (e.g., 84,819 packet/sec) [42].

For the simulation setup, we analyzed performance characteristics of each processor core in terms of the power dissipation and execution time. We relied on detailed gate-level realization of a 32bit RISC-type processor compatible with [43] in TSMC 65nmLP library in order to accurately evaluate the power dissipation of a core. By varying the voltage and frequency values during the simulation, we achieved power and delay numbers with Power Compiler [44] for the core after running the same tasks. Furthermore, we utilized a back-annotated SAIF (Switching Activity Interchange File), which captures switching activity factor with test patterns, based on the RTL simulation to achieve accurate power numbers. For simplicity, we defined a set of four actions, i.e., a_0 = [0V, 0Hz], a_1 = [1.00V, 150MHz], a_2 = [1.08V, 200MHz], and a_3 = [1.20V, 250MHz], assuming that the voltage of the core is determined based on the operating frequency. Note that a_0 is used to indicate a power-off (power gating) state in which high Vth sleep transistors are used to disconnect the circtuit power supply from logic gates when the circuit becomes inactive.

B. Detailed Results

In the first experiment, we generated a training set by running a set of tasks on the processor core as follows. First, we considered a scenario whereby the core accepts two types of tasks: low-priority and high-priority, where a high-priority task can move ahead of all low-priority tasks waiting in the queue. Next, we defined a set of input features {type of task, occupancy state of the SQ, arrival rate of task} and output measures {power dissipation [mW], execution time [ns]},

[1] In the current world of high-speed networking, where multiple processing cores reside within a single system, the ability of the networking protocol stack of the operating system to scale well on a multi-core system is inhibited because the architecture of conventional Network Driver Interface Specification (NDIS 5.1 and earlier versions) limits receive protocol processing to a single core. Receive-Side Scaling (RSS) resolves this issue by allowing the network load from a network adapter to be balanced across multiple cores.

similar to TABLE I. During the training phase, voltage and frequency values are assigned to the processor core based on simple requirements such as:

- The core runs faster when high-priority tasks with medium or high arrival rates arrive under low or medium queue occupancy,
- The core runs slower when low-priority tasks with low or medium arrival rates arrive under medium or high queue occupancy.

Fig. 6 shows various input features during the training phase, whereas Fig. 7 depicts the corresponding output measures for 100 training sets. Note that profiling output measure (e.g., power dissipation) at runtime is feasible with support of specific hardware such as external current sensors or internal architectural counters for each core. An external current sensor [46], supplied by a voltage regulator which also provides power to the corresponding core, enables online current measurement, which is accumulated in the current accumulator, digitally multiplied by voltage value, and fed into a power dissipation accumulator. On the other hand, internal architectural counters used to compute the power consumed by cores count a number of relevant events and appropriately weight the counter values. For example, the total numbers of load/store instructions, arithmetic/logic instructions, floating-point operations, and retirement executions for each core are counted and summed up after being multiplied by appropriate weights [47].

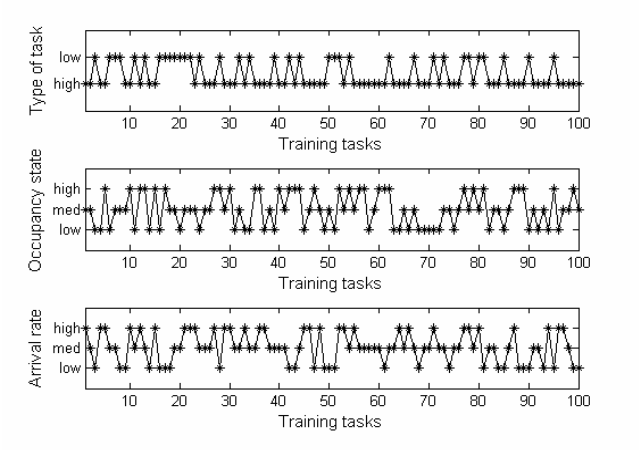


Fig. 6. Input features during training phase.

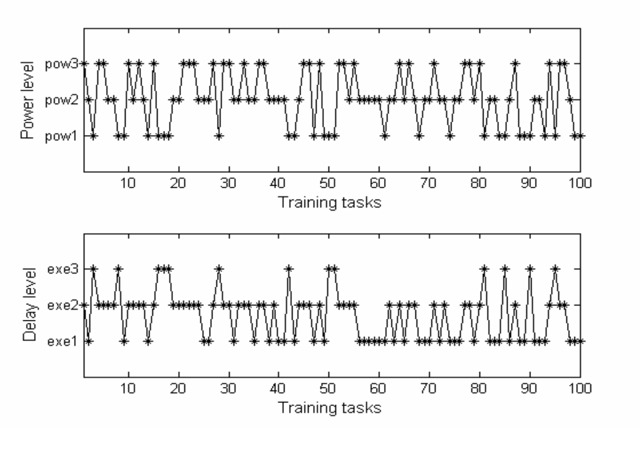


Fig. 7. Output measures during training phase.

The decision boundaries for an output measure are obtained as follows. First, we assign various labels to the input features based on our simulation results. After running a number of simulations, we derive probabilistic density functions for the power consumption of the core (cf. Fig. 8) for three observations: $o_1 = N(35.8, 2.2)$, $o_2 = N(44.2, 3)$, and $o_3 = N(50.5, 1.8)$. Next, the two separation points between neighboring observations are calculated as: δ_1 = 39.4 and δ_2 = 48.1. The minimum power (30.3mW) and maximum power (56.0mW) consumption values for active mode of the processor core operation are used as the lower and upper bounds of the power dissipation range. The decision boundaries for the execution delay are also obtained in a similar manner. Consequently, the classes of output measures are defined according to TABLE III.

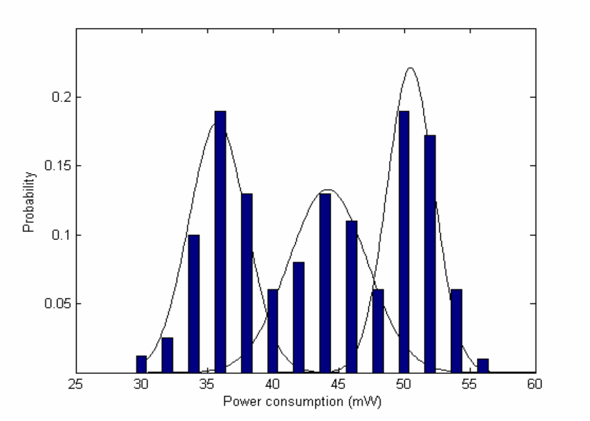


Fig. 8. Probability density functions for power dissipation.

TABLE III
CLASSES OF OUTPUT MEASURES

Power dissipation (mW)			Execution time (ns)		
pow_1	pow_2	pow_3	exe_1	exe_2	exe_3
[30.0 39.4]	(39.4 48.1]	(48.1 56.0]	[14.1 21.5]	(21.5 28.5]	(28.5 35.7]

To ensure high accuracy in classification, we define classification error [48] as follows. The error in classification is calculated as

$$ER = \int L(f(x), y) Prob(x, y)\, dx\, dy \tag{17}$$

where $f(x)$ denotes the predicted output measure while y is the actual output measure. L(.,.) is a general loss function. For our target application, we use a 0-1 loss function, i.e.,

$$L(f(x), y) = \begin{cases} 0 & if\ \ y = f(x) \\ 1 & otherwise \end{cases} \tag{18}$$

where $f(x) = \arg\max_{Y} Prob(Y \mid X = x)$ in this case. The class-conditional classification accuracy is then given by 1 – ER. It is a measure of the performance of the classifier. Considering the input feature that we used as an example in section III, the accuracy reaches around 88% in classification. In addition, the training set size can greatly impact the classification accuracy, so we performed simulations to determine an appropriate size by varying the set size from 50 to 3000 as shown in Fig. 9. We have thus empirically determined that a training set size of 1000 is adequate. Note that substantial reductions in training set size may be possible if interest is focused on a single class (e.g., only power dissipation) [49].

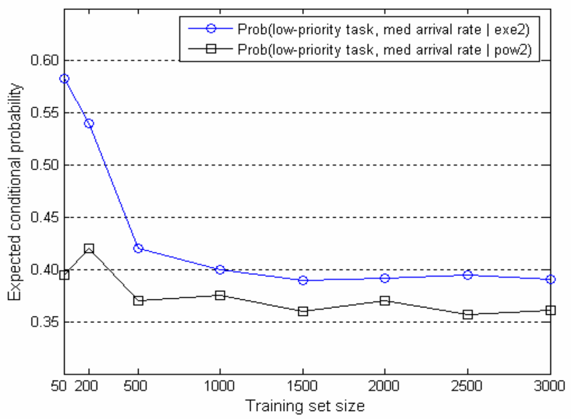


Fig. 9. Selection of the training set size.

TABLE IV
DIFFERENT CHARACTERISTICS OF TRAINING SETS

	Input feature				
	Priority		Arrival rate		
	High	Low	High	Med	Low
Training set A	50%	50%	20%	60%	20%
Training set B	20%	80%	20%	20%	60%
Training set C	80%	20%	60%	20%	20%

It is worthwhile to consider a scenario whereby the characteristics of the task may change over time [51][52]. If the workload characteristics change over time, the performance of the classification can degrade. This is because, having relied on biased input features during the training phase, the classifier may not be able to correctly predict the output measure class of a given input feature. For example, consider different sets of training data as shown in TABLE IV. Suppose we train three classifiers based on training set A, set B, or set C. Next we randomly generated 100 tasks and perform classification for each incoming task, followed by an optimal action for each task based on the classification result.

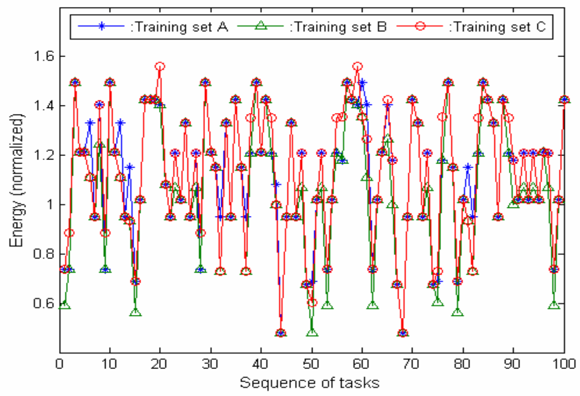


Fig. 10. Comparison of energy dissipations, where actions are commanded by a classifier based on different training sets.

Fig. 10 shows the normalized energy dissipation by the issued actions commanded by the three aforesaid classifiers. The results are quite different for the three classifiers; this shows the importance of using a representative training set.

To validate the above statement, we considered a scenario whereby a classifier is trained based on some expected input characteristics but is subsequently used to classify input features with different characteristics. In particular, we first trained a classifier with training set B and used it to determine the output measure class of elements in set C (modeling the case whereby the input characteristics changed over time from those of set B to those of set C). Fig. 11 shows the comparison in energy dissipation for 100 tasks between this case and one in which a classifier ("with update") was trained based on set C and then ran on data with similar characteristics as those of set C. It is clearly seen that the classifier "with update" outperforms that "without update". Finally, notice that we could have trained a better classifier by using data from all three training sets A, B, and C. TABLE V shows the normalized total energy dissipation for 100 tasks by various classifiers, where each classifier is trained with the specific training set. It is clearly seen that the classifier trained with all training sets consumes less energy, compared to other classifiers.

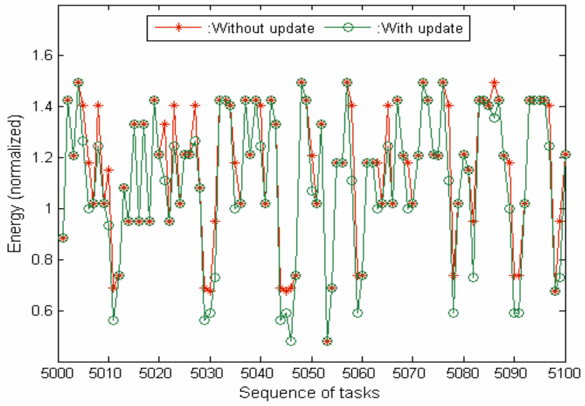


Fig. 11. Evaluation of energy dissipation for a given scenario.

TABLE V
NORMALIZED TOTAL ENERGY DISSIPATION FOR VARIOUS CLASSIFIERS

	Training sets						
	A, B, C	A, B	A, C	B, C	A	B	C
Energy	108.2	113.7	115.3	114.3	125.0	122.6	127.5

Next we investigated the energy-efficiency of the proposed DPM technique by comparing it with i) the Stochastic PM technique of [54] which uses a stochastic optimization approach for power control based on the service request rates and ii) the Global PM technique of [55] which uses a feedback mechanism to sense per-core power and performance states. The cost function of [54] is the power-delay product, which makes the comparison easy. For simplicity, the waiting time at the queue was considered to be fixed. To do a fair comparison, the power and performance states of [55] were represented by the power and delay levels defined in our experimental setup. Here we assumed that the latency overhead of DVFS is on the order of several tens of microseconds. We used four VF values where $a_0 < a_1 < a_2 < a_3$, with a_0 corresponding to a power-off state, a_1 denoting the lowest (operational) power and performance state, and a_3 denoting the highest power and performance state.

Stochastic PM technique: It employs a DPM assignment strategy such that a power manager is triggered to issue a DVFS command based on a precomputed and stored policy table. The key into this hash table is the current state of the system which is a pair representing the current power-performance state of the processor and the request arrival rate. The policy table itself is computed off-line using the stochastic optimization framework of [54] where the objective is total expected system power and the constraint is an upper bound on delay. In our simulations, the state transition (i.e., power mode transition) probabilities are calculated from offline simulations. For example, when the system state changes into the lowest power mode (e.g., pow_1) the power manager assigns a command as follows:

if the processor is idle (i.e., task arrival rate = 0), a_0
else if $0 \leq$ arrival rate of tasks ≤ 0.33, a_1
else if $0.33 <$ arrival rate of tasks ≤ 0.67, a_2
else if $0.67 <$ arrival rate of tasks < 1, a_3.

Global PM technique: It employs a feedback-control DPM strategy based on the Priority policy of [55] where the power manager assigns different priorities to different tasks so as to meet a specific global power budget by adjusting power modes of individual processing cores. Similar to [55], processor core 4 has the highest priority (will run as fast as possible) while processor core 1 has the lowest priority (will be the first to slow down in case of a power budget overshoot).

Bayesian PM: Employ the Bayesian learning-based DPM method described in this paper.

We generated a number of tasks by selecting the arrival rate of tasks randomly, where $0 \leq$ the arrival rate of tasks < 1, and applied the above-mentioned DPM policies to the processor.

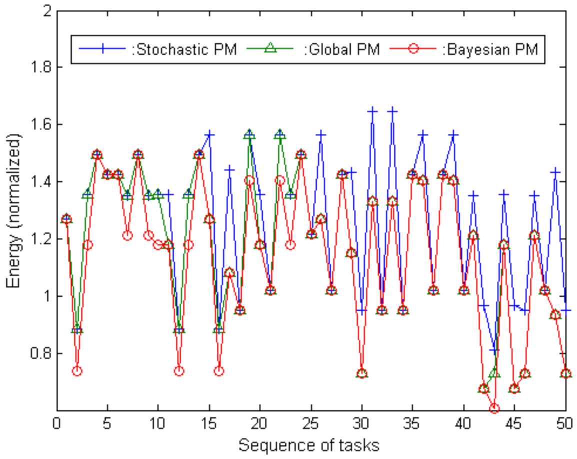


Fig. 12. Energy dissipation comparison for a given experimental setup.

The simulation results in Fig. 12, which report the (normalized) energy dissipation of each task (numbered from 1 to 50) for the processor, show that the proposed DPM technique, i.e., Bayesian PM exhibits sizeable energy savings up to 24% and 15% (on average) compared to [54] and [55], respectively. Considering the performance of the processor, the overhead of performing classification in the Bayesian PM 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 any VF change).

Experimental results in TABLE VI, which also reports the characteristics of the workload distribution for each processor (e.g., Proc1 receives 50 tasks whereas Proc4 receives 200 tasks), demonstrate that, compared to the Stochastic and Global PM policies, the proposed Bayesian classification–based power management policy achieves system-wide energy (normalized) savings of up to 20.5% and 11.5% (these are the normalized averages on four processors when α = 1), respectively. It is also seen that if we consider the missing input feature (e.g., α = 0.95 and α = 0.90), there is little performance degradation.

TABLE VI
ENERGY SAVINGS IN THE MPS

Processor	Number of tasks	Energy (Stochastic)	Energy (Global)	Bayesian		Energy saving over	
				Prob. (α)	Energy	Stochastic	Global
Proc1	50	61.0	54.2	1.00	46.8	23.1%	13.6%
				0.95	49.2	19.3%	9.2%
				0.90	50.8	16.7%	6.3%
Proc2	100	121.5	109.1	1.00	97.8	19.5%	10.3%
				0.95	98.2	19.1%	9.9%
				0.90	99.8	17.83%	8.5%
Proc3	150	189.2	170.6	1.00	150.6	20.4%	11.7%
				0.95	154.3	18.4%	9.5%
				0.90	156.2	17.4%	8.4%
Proc4	200	243.4	220.0	1.00	197.6	18.8%	10.2%
				0.95	203.5	16.3%	7.5%
				0.90	205.4	15.6%	6.6%

VIII. CONCLUSION

The 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 MPS, 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 have demonstrated that the proposed DPM framework results in significant energy savings for various workloads in MPSs.

REFERENCES

[1] D. I, Q. Xie, and P.H. Chou, “Scalable Modeling and Optimization of Mode Transitions based on Decoupled Power Management Architecture,” *Proc. of Design Automation Conference*, Jun. 2003, pp. 119-124.

[2] Y-H. Lu and G. De. Micheli, "Comparing System-Level Power Management Policies," *IEEE Design & Test of Computers*, Vol. 18, Issue 2, pp. 10-19, Mar-Apr. 2001.

[3] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar., "Power and Thermal Management in Intel Core Duo Processor," *Intel Technology Journal*, Vol. 10, Issue 2, pp. 109-122, May 2006.

[4] H. Jung and M. Pedram, "Continuous Frequency Adjustment Technique based on Dynamic Workload Prediction," *Proc. of International Conference on VLSI Design*, Jan. 2008, pp.415-420.

[5] A. Iyer and D. Marculescu, "Power Efficiency of Voltage Scaling in Multiple Clock, Multiple Voltage Cores," *Proc. of International Conference on Computer Aided Design*, Nov. 2002, pp. 379-386.

[6] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen., "Single ISA Heterogeneous Multicore Architecture: The Potential for Processor Power Reduction," *Proc. of Symposium on Microarchitecture*, Dec. 2003, pp. 81-93.

[7] 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 Symposium on High-Performance Computer Architecture*, Feb. 2005, pp. 178-189.

[8] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi, " Profile-based Optimization of Power Performance by using Dynamic Voltage Scaling of a PC cluster," *Proc. of Parallel and Distributed Processing Symposium*, Apr. 2006, pp. 8-16.

[9] B. Mochocki, D. Rajan, X.S. Hu, C. Poellabacer, K. Otten, and T. Chantem, "Network-Aware Dynamic Voltage and Frequency Scaling," *Proc. of Real Time and Embedded Technology and Application Symposium*, Apr. 2007, pp. 215-224.

[10] E. Chung, L. Benini, and G. De. Micheli, "Dynamic Power Management Using Adaptive Learning Tree," *Proc. of International Conference on Computer Aided Design*, Nov. 1999, pp. 274-279.

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

[12] S. Ma and C. Ji, "Performance and Efficiency: Recent Advances in Supervised Learning," *Proc. of IEEE*, Vol. 87, No. 9, pp.1519 – 1535, Sep. 1999.

[13] P. Teich, "Multi-Core Processor Technology: Maximizing CPU Performance in a Power-Constrained World," *presentation slide of Microsoft Windows Hardware Engineering Conference*, Apr. 2005.

[14] H. Zhong, S. A. Lieberman, and S. A. Mahke, "Extending Multicore Architectures to Exploit Hybrid Parallelism in Single-Thread Applications," *Proc. of Int'l Symposium on High Performance Computer Architecture*, Mar. 2007, pp.25-36.

[15] V. Paxson, R. Sommer, and N. Weaver, "An Architecture for Exploiting Multi-Core Processors to Parallelize Network Intrusion Prevention," *Proc. of .IEEE Sarnoff Symposium*, Apr. 2007, pp.1-7.

[16] T.D. Burd and R.W. Brodersen, "Design Issues for Dynamic Voltage Scaling," *Proc. of International Symposium on Low Power Electronics and Design*, Aug. 2000, pp. 9-14.

[17] A. Mittal and A. Kassim, *Bayesian Network Technologies: Applications and Graphical Models*, IGI Publishing, 2007.

[18] T. Mitchell, *Machine Learning*, McGraw Hill, 1997.

[19] K. Huang, Z. Xu, I. King, M. R. Lyu, Z. Zhou, "A Novel Discriminative Naive Bayesian Network for classification," In *Bayesian Network Technologies: Applications and Graphical Models*. Ankush Mittal, Ashraf Kassim, Tele Tan (Eds.), March, 2007, pp 1-12, Idea Group Inc.

[20] Vapnik, V.N, *The nature of statistical learning theory* (2nd ed.). New York: Springer-Verlag. 1999.

[21] G. Theocharous, S. Mannor, N. Shah, P. Gandhi, B. Kveton, S. Siddiqi, and C. Yu, "Machine Learning for Adaptive Power Management," *Intel Technology Journal*, Vol. 10, Issue 4, pp.299 – 310, Jul. 2006.

[22] G. Dhiman, and T. S. Rosing, "Dynamic Power Management Using Machine Learning," *Proc. of Int'l Conference on Computer Aided Design*, Nov. 2006, pp. 747-754.

[23] G. Dhiman, and T. S. Rosing, "Dynamic voltage frequency scaling for multi-tasking systems using online learning," *Proc. of International Symposium on Low Power Electronics and Design*, Jul. 2007, pp.207-212.

[24] A. Weissel, and F. Bellosa, "Self-Learning Hard Disk Power Management for Mobile Devices," *Proc. of Int'l Workshop on Software Support for Portable Storage*, Oct. 2006, pp. 33 – 40.

[25] C. Rusu, N. AbouGhazaleh, A. Ferreira, R. Xu, B. Childers, R. Melhem, and D. Mosse, "Integrated CPU and L2 cache Frequency/Voltage Scaling using Supervised Learning," *Proc. of Workshop on Statistical and Machine Learning Approaches applied to Architectures and Compilation*, Jul. 2007, pp. 41 – 50.

[26] W. Cohen, "Fast Effective Rule Induction," *Proc. of 12th Int'l Conference on Machine Learning*, Dec. 1995. pp. 115-123.

[27] R. Quinlan, C4.5: *Program for Machine Learning*, Morgan Kaufmann Publisher, 1993.

[28] A. Statnikov, C.F. Aliferis, I. Tsamardinos, D.P. Hardin, and S. Levy*, A Comprehensive Evaluation of Multicategory Classification Methods for Microarray Gene Expression Cancer Diagnosis*, Bioinformatics, 2004.

[29] C. Cortes and V. Vapnik, "Support-Vector Networks," *Journal of Machine Learning*, Vol. 20, No. 3, pp. 273-297, 1995.

[30] R. Knauerhase, P. Brett, T. Li, B. Hohlt, and S. Hahn, "Using OS Observations to Improve Performance in Multi-core Systems, " *Proc. of IEEE Micro*, Vol. 28, Issue 3, pp. 54-66, May-Jun. 2008.

[31] L. Zhao, R. Iyer, R. Illikkal, J. Moses, S. Makineni, and D. Newell, "CacheScouts: Fine-Grain Monitoring of Shared Caches in CMP Platforms," *Proc. of Int'l Conference on Parallel Architecture and Compilation Techniques*, Oct. 2007, pp. 339-349.

[32] M.L. Seltzer, B. Raj, and R.M. Stern, "A Bayesian classifier for spectrographic mask estimation for missing feature speech recognition," *Journal of Speech Communication*, Vol. 43, pp. 379-393, Mar. 2004.

[33] A. Dempster, N. Laird, and D. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society,* Series B, 39(1), pp. 1-38, 1977.

[34] J.A. Bilmes, "A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Model," *Technical Report*, TR-97-021, U.C. Berkeley, 1998.

[35] R. A. Fisher, *Statistical Methods and Scientific Inference*, Macmillan Pub Co. 1973.

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

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

[38] 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.

[39] Ken Salchow, "Load Balancing 101," white papers from F5 Inc., http://www.f5.com/pdf/white-papers/evolution-adc-wp.pdf and http://www.theacademy.ca/load-balancing101-wp.pdf .

[40] White paper, "Scalable Networking: Eliminating the receive processing bottleneck – Introduction RSS," WinHEC 2004 version, Apr. 2004 http://www.microsoft.com/whdc/.

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

[42] H. Jung, and M. Pedram, "A Unified Framework for System-level Design: Modeling and Performance Optimization of Scalable Networking Systems," *Proc. of Int'l Symposium on Quality of Electronic Designs*, Mar. 2007, pp.198-203.

[43] OpenRISC processor. http://www.opencores.org. Opencore, 2009.

[44] Synopsys compiler. http://www.synopsys.com. Synopsys, 2009.

[45] B. Calhoun, J. Kao, and A. Chandrakasan, *Leakage in Nanometer CMOS Technologies*, Springer, 2006.

[46] White paper, "Bi-directional current/power monitor with I^2C Interface," Sep. 2008, http://focus.ti.com.

[47] V. Srinivasan, D. Brooks, M. Gschwind, and P. Bose, "Optimizing Pipelines for Power and Performance," *Proc. of International Symposium on Microarchitecture,* Nov. 2002, pp.333-344.

[48] V. Pronk, S.V.R. Gutta, and W.F.J. Verhaegh, "Incorporating Confidence in a Naïve Bayesian Classifier," *Lecture Notes in Computer Science: User Modeling 2005*, pp.317-326, Aug. 2005.

[49] G.M. Foody, A. Mathur, C. Sanchez-Hernandez, and D. S. Boyd, “Training set size requirements for the classification of a specific class,” *Journal of remote sensing of environment*, Vol. 104, Issue 1, pp. 1 – 14, Sep. 2006.

[50] C. McNairy and R. Bhaita, “Montecito: A Dual-Core, Dual-Thread Itanium Processor,” *IEEE Micro*, Vol. 25, Issue 2, pp. 10-20, Mar-Apr., 2005.

[51] S. Chaudhuri and V. Narasayya, “An Efficient Cost-driven Index Tuning Wizard for Microsoft SQL Server,” *Proc. of 23rd International Conference on Very Large Databases*, Sep. 1997, pp. 146-155.

[52] I. Ahmand, “Easy and Efficient Disk I/O Workload Characterization in VMware ESX Server,” *Proc. of International Symposium on Workload Characterization*, Sep. 2007, pp. 149-158.

[53] E. Castro-Leon, S. Nayak, and D. Shenkar, “Data Center Power and Thermal Management using Intel Data Center Manager Software Development Kit”, http://software.intel.com. Jul. 2009.

[54] Q. Qiu and M. Pedram, “Dynamic Power Management based n Continuous Time Markov Decision Process,” *Proc. of Design Automation Conference*, Jun. 1999, pp.555-561.

[55] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose and M. Martonosi, “An Analysis of Efficient Multi-Core Power Management Policies: Maximizing Performance for a Given Power Budget,” *Proc. of Int’l Symposium on Microarchitecture,* 2006, pp.347-358