Workload Uncertainty Characterization and Adaptive Frequency Scaling for Energy Minimization of Embedded Systems

Anup Das[‡†], Akash Kumar[†], Bharadwaj Veeravalli[†], Rishad Shafik[‡], Geoff Merrett[‡], Bashir Al-Hashimi[‡]
[‡]School of ECS, University of Southampton and [†]Department of ECE, National University of Singapore
[‡]{a.k.das,rishad.shafik,gvm,bmah}@ecs.soton.ac.uk and [†]{akdas,akash,elebv}@nus.edu.sg

***Abstract*—A primary design optimization objective for multicore embedded systems is to minimize the energy consumption of applications while satisfying their performance requirement. A system-level approach to this problem is to scale the frequency of the processing cores based on the readings obtained from the hardware performance monitors. However, performance monitor readings contain uncertainty, which becomes prominent when applications are executed in a multicore environment. This uncertainty can be attributed to factors such as cache contention and DRAM access time, that are very difficult to predict dynamically. We demonstrate that such uncertainty can be controlled to make better decision on the processor frequency in order to minimize energy consumption. To achieve this, we propose a multinomial logistic regression model, which combines probabilistic interpretation with maximum likelihood (ML) estimation to classify an incoming workload, at run-time, into a finite set of classes. Every workload class corresponds to a frequency pre-determined using an appropriate training set and results in minimum energy consumption. The classifier incorporates (1) uncertainty with arbitrary probability distribution to estimate the actual frame workload; and (2) the frequency switching overhead, neither of which are considered in any of the existing approaches. The classified frequency is applied on the processing cores to execute the workload. The proposed approach is engineered into an embedded multicore system and is validated with a set of standard multimedia applications. Results demonstrate that the proposed approach minimizes energy consumption by an average 20% as compared to the existing techniques.**

I. Introduction

Multimedia applications, such as video encoding and decoding, are characterized by different execution phases, which are defined as a group of consecutive frames. The average workload of the frames comprising a phase (inter) varies significantly across the different phases; however, the workload variation within each phase (intra) is relatively low. Proactive energy management involves predicting these dynamic workloads apriori to determine the most appropriate frequency for every phase such that performance constraint is satisfied while minimizing the energy consumption [1]. Studies have been conducted recently to use machine learning to determine the minimum frequency through continuous feedback from the hardware performance monitoring unit (PMU) [2]–[10]. These approaches suffer from the following limitations.

First, some of the practical aspects of multicore systems are ignored in the existing works. Specifically, the CPU cycle count for a frame, obtained by reading the PMU registers at run-time, is assumed to be a true indicator of the frame workload. However, as we show in this paper, the PMU register readings contain a certain amount of uncertainty, influenced by factors such as cache contention, DRAM access, etc., that can have a significant impact on energy savings. This uncertainty is difficult to estimate at run-time due to the unpredictability associated with these factors, especially for multicore systems with a realistic assumption of concurrently executing routine applications. Thus, although workload estimation based on CPU cycle count leads to energy savings using DVFS, a significant improvement is possible by estimating the uncertainty as we show in this paper. Second, the existing approaches do not consider voltage and frequency switching overhead, which is significant in modern multicore systems. Last, the classical workload predication-based energy minimization techniques work in an ad hoc manner by predicting the workload and deciding the frequency based on this predicted workload. On the other hand, workload history-based statistical classification approaches determine the probability that a sudden spike (positive or negative) in the workload is due to a change in the phase of the workload, that needs to be processed at a different frequency. Thus, instead of acting instantaneously, the classifier evaluates the probability distribution of the different classes based on the workload change and the most probable frequency is selected such that the scaling leads to energy reduction for future workloads. However, these classifiers require characterization using training data, forming them to be part of a supervised learning algorithm. Modern multicore operating systems, such as Linux and Android, also support dynamic frequency scaling during application execution. The default and the most popular *ondemand* power governor [11] uses the current workload to determine the voltage-frequency value to process the future workload (a reactive approach). As we show in this paper, the energy reduction using the *ondemand* power governor can be outperformed using a naive predictive heuristic.

To address the above limitations, we propose a multinomial logistic regression-based classification technique, that classifies the workload at run-time, into a fixed set of classes. To accomplish this, the classifier determines the probability distribution of these classes that changes with the input sample (i.e., a window of prior frame workloads). The parameters of the classifier are determined using maximum likelihood estimation using a training set. The maximum likelihood estimation incorporates the uncertainty associated with unmodeled factors, with arbitrary probability distributions, and determines the frequency considering the frequency switching overhead.

Contributions: Following are our key contributions:

- a frame-history based statistical workload classification approach for dynamic energy minimization;
- incorporating the unmodeled factors with arbitrary probability distribution in frame processing; and
- considering frequency switching overhead for energy optimization.

The proposed approach is engineered on a multicore embedded platform running Linux. Experiments conducted with multimedia applications demonstrate that the proposed approach minimizes energy consumption by an average 20% with no degradation of performance.

To the best of our knowledge, this is the first work that models the unpredictability in estimating the frame workload, to accurately predict the frequency of operation. The rest of this paper is organized as follows. An overview of the related works is provided in Section II. This is followed by the uncertainty in workload estimation in Section III and the design methodology in Section IV. The background on classification and its integration in the overall methodology are provided in Sections V and VI, respectively. Results are discussed in Section VII and the paper is concluded in Section VIII.

II. Related Works

Dynamic energy minimization has attracted significant attention both in industry and in academia [12]. Continuous frequency adjustment technique is proposed in [1] based on predicted workload, which is formulated as an initial value problem (IVP). The technique in [2]–[5] uses online learning to

978-3-9815370-4-8/DATE15/©2015 EDAA

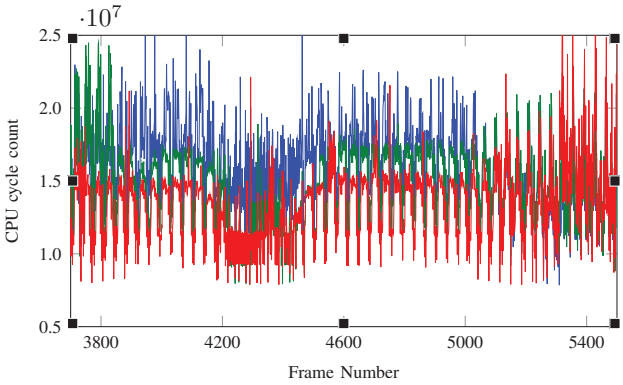


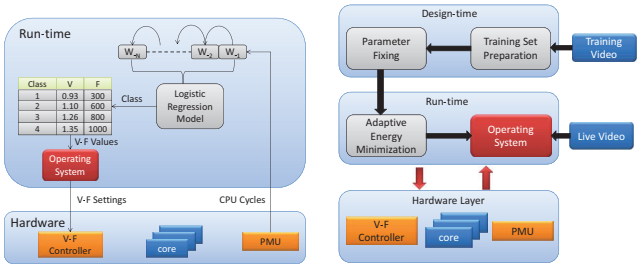
Fig. 1: CPU cycle count from consecutive runs.

select the most appropriate frequency for the processing cores based on the workload characteristic of a given application. A workload characteristics aware thread scheduler is proposed in [6] based on dynamic workload characterization. In [7], a supervised learning in the form of a Bayesian classifier for energy management is proposed. This framework learns to predict the system performance from the occupancy state of the global service queue. The predicted performance is then used to select the frequency from a pre-computed policy table. In [8], [9], a multinominal logistic regression classifier is built using a large volume of performance counters by offline workload characterization. This classifier is queried at run-time for a given application to predict the workload, and select the frequency and thread packing such that performance is maximized under a given power cap. A workload aware approach is proposed in [10] based on control theoretic principles. Our proposed approach differs from these techniques by addressing the three limitations discussed in Section I. Specifically, the input features of the proposed classifier is composed of the CPU cycle counts of a window of prior frames to determine the probability distribution of switching workload class, rather than an ad-hoc optimization using the CPU statistics of the current frame only. Moreover, by introducing the modified maximum likelihood estimation for parameter fitting, the uncertainty in workload estimation and voltage and frequency switching overhead are both minimized.

III. Uncertainty in Workload Estimation

For energy optimization, the minimum frequency for an application is determined based on the CPU cycle count (henceforth referred to as workload) read from the PMU registers. The underlying assumption is that the CPU cycle count corresponds to frame processing only. In modern systems, there are a number of applications that continue to operate in the background. Some of these applications are user controlled such as web page rendering, email checking and virus scanning. There are also system-related applications that are routinely executed on the processing cores. Some of these applications are beyond the knowledge of the users and cannot be forcefully exited. As a result, the PMU register readings are not a true indicator of the actual frame workload. This can be easily verified by executing the same video multiple times and recording the CPU cycle count for every run. This is shown in Figure 1 for three consecutive runs (shown in red, green and blue) without any change in the operating environment (i.e. with the same set of background applications).

As can be seen, the PMU readings differ across the three observations. The PMU readings for the observation plotted in green are higher than the other two readings at around 3,800 frames; the readings for the observation shown in red are higher at around 5400 frames; and the readings for observation shown in blue are higher at around 4,200 frames. Thus, although the workload trends are the same, all the PMU readings vary across the three runs with variations as high as 60% for some of the frames. Clearly, the observed readings from the PMU registers for a frame contain uncertainty that



(a) Adaptive energy minimization (b) Design methodology

Fig. 2: Proposed approach.

needs to be estimated in order to accurately determine the minimum frequency. This problem is more severe in a multicore environment due to the unpredictability associated with some of the factors such as cache contention or external memory access time. These unmodeled factors do not follow a known probability distribution. To estimate the impact of workload uncertainty on the frequency value, let $\tilde{w}$ and w denote the observed and the actual frame workloads, respectively, with $\tilde{w} = w + e$, where e is the workload uncertainty. The observed and the required workload frequencies are related according to

$$\frac{f_{required}}{f_{observed}} = \frac{\tilde{w} - e}{\tilde{w}} \leq 1 \tag{1}$$

Clearly, estimating the uncertainty in the observed workload leads to further scope for energy improvement. Thus, the problem we are addressing is as follows. Given the workload obtained from the PMU registers at run-time, how to estimate the workload uncertainty (e), being agnostic of its probability distribution, such that the voltage-frequency value corresponding to the actual workload w can be applied on the system.

IV. Design Methodology

Figure 2a and 2b shows respectively the adaptive energy minimization approach and the classification-based design methodology based on this approach. An overview is provided on the interaction of the different blocks of this methodology.
Application: Typically, multimedia applications are characterized with a performance constraint specified as *frames per second*, reciprocal of which gives the timing constraint for processing a frame. The application source code is annotated to include this timing requirement.
Operating System: The operating system is responsible for coordinating the application execution on the hardware. After processing every frame of an application, the operating system stalls execution and triggers the classifier, which predicts the class for the next frame. This class is translated to a frequency value for the CPU cores. The operating system applies this frequency on the CPU cores using the `cpufreq` utility.
Hardware: The hardware consists of processing cores with a performance monitoring unit to record performance statistics. Of the different performance statistics available, we focus on CPU cycle count. After processing every frame, the PMU readings are collected using the `perfmon` utility. Subsequently, the readings are reset to allow recording for the next frame. Finally, before the start of the next frame, the frequency value set by the operating system is first converted to a corresponding CPU clock divider setting and is then written into appropriate CPU registers. The frequency is scaled to execute the next frame.

V. Background on Classification

Statistical classification is the process of identifying a class (from a set of discrete classes) for a new observation, based on a training set of observations, whose class is known apriori [13]. In this work we focus on a discriminative classifier – logistic regression applied to a multinomial variable [14]. This type of classifier predicts the probability distribution over a set of classes from a sample input to learn a direct mapping from the input sample to the output class. The logistic regression based classification is composed of two steps – modeling to estimate the probability distribution of the different classes for a given input, and parameter fitting to estimate the parameters of the logistic regression model. These are described next.

A. Multinomial Logistic Regression Model

Assumptions:
$\mathcal{A}_1$: There are K discrete frequencies supported by the hardware. The incoming workload is assigned to one of these values, depending on which frequency results in the least energy consumption while satisfying the performance requirement. This is same as classifying into one of K classes.
$\mathcal{A}_2$: The class of the next video frame is predicted based on the workloads of the N previous frames. These are identified by $X = \begin{pmatrix} x_1 & x_2 & \cdots & x_N \end{pmatrix} \in \mathbb{R}^{1\times N}$, where x_i is the workload of the i^{th} previous frame.
$\mathcal{A}_3$: The workload class is denoted by the variable $y \in [1, 2, \cdots, K]$ and the logistic regression model is represented by the hypothesis h_θ, with parameter $\theta \in \mathbb{R}^{(K-1)\times N}$.

It can be shown that for a given input *feature* set X, the logistic regression model outputs $h_\theta(X)$ given by

$$h_\theta(X) = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{K-1} \end{bmatrix} = \begin{pmatrix} \frac{e^{\left(\theta^{(1)^T}\cdot X\right)}}{\sum_{j=1}^{K} e^{\left(\theta^{(j)^T}\cdot X\right)}} & \cdots & \frac{e^{\left(\theta^{(K-1)^T}\cdot X\right)}}{\sum_{j=1}^{K} e^{\left(\theta^{(j)^T}\cdot X\right)}} \end{pmatrix} \quad (2)$$

The output class y is given by

$$y = \underset{l}{\operatorname{argmax}} \{p_l \ \forall\, l \in [1, 2, \cdots, K]\} \quad (3)$$

B. Maximum Likelihood Estimation

We consider a training set of M samples generated independently and identically. For each of these samples, the input *feature* X and the output class y are known apriori and the input-output pairs are identified as $(X^{(i)}, y^{(i)})\ \forall i \in [1, 2, \cdots, M]$. The maximum likelihood estimation is a technique to estimate the parameters (θ in our case) of a model by maximizing the likelihood of the joint probability distribution of the different observations. This is given by

$$\ell(\theta) = \ln\left(\mathcal{L}(\theta)\right) = \sum_{i=1}^{M}\sum_{l=1}^{K} \mathcal{I}(y^{(i)} = l) \cdot \ln\left(\frac{e^{\left(\theta^{(l)^T}\cdot X^{(i)}\right)}}{\sum_{j=1}^{K} e^{\left(\theta^{(j)^T}\cdot X^{(i)}\right)}}\right) \quad (4)$$

C. Uncertainty Interpretation

First we define two new terms – observed class and actual class. The observed class (denoted by $\tilde{y}$) is the class perceived at the output of the logistic regression model corresponding to input X and includes the uncertainty. Let the variable y denote the actual class as before. Using the basic principles of probability theory, the probability of the observed class is

$$P(\tilde{y} = i \mid X) = \sum_{r=1}^{K} P(\tilde{y} = i \mid y = r) \cdot P(y = r \mid X) = \sum_{r=1}^{K} \gamma_{i,r} \cdot p_r \quad (5)$$

where $\gamma_{i,r}$ is the probability that the actual class r is flipped to the observed class i. Using this probability and the definition of the likelihood function, the log likelihood function is[1]

$$\ell(\theta, \gamma) = \sum_{i=1}^{M}\sum_{l=1}^{K} \mathcal{I}(\tilde{y}^{(i)} = l) \ln\left(\sum_{r=1}^{K} \gamma_{l,r} \cdot p_r\right) \quad (6)$$

Finally, the output of the hypothesis is modified as

$$h_\theta(X) = \begin{bmatrix} \left(\sum_{r=1}^{K} \gamma_{1,r} \cdot p_r\right) & \cdots & \left(\sum_{r=1}^{K} \gamma_{K-1,r} \cdot p_r\right) \end{bmatrix}^T \quad (7)$$

VI. Implementation Details

As mentioned in Section V, there are two components of the logistic regression based classification approach – multinomial logistic regression model and parameter fitting using maximum likelihood estimation. The multinomial logistic regression model works at run-time and is used to classify workloads with uncertainty for energy minimization. The maximum likelihood estimation works on training set to determine the parameters of the model in order to improve the efficiency of the probabilistic classification. A point to be noted here is that, the parameter fitting process is a one-time overhead and can be performed both offline (at design-time) or online (at run-time). Different components of the proposed adaptive energy minimization approach are described next.

[1]The derivation steps are omitted for space limitation.

ALGORITHM 1: Training set preparation (video processing demonstration).

```
Input: Video database vDB, set of K voltage-frequency pairs
       {(V_j, F_j) ∀j ∈ [1, 2, ··· , K]}, timing constraint for frame
       processing t_c and frame window N
Output: Training set database tsDB
1  Initialize arrays t.clear(), n.clear() and s.clear();
2  for i = 1 to N_v do
3      curr_video = vDB(i);
       // Learning section of the algorithm
4      for j = 1 to K do
5          cpufreq-set -f F_j;
6          for w = 1 to #(curr_video.frame) do
7              perf-event.start(n(w),CPU_CYCLES);
8              start_time = current_time();
9              process(curr_video.frame(w));
10             t(j, w) = current_time() - start_time;
11             perf-event.stop();
12         end
13     end
14     for w = 1 to #(curr_video.frame) do
15         s(w) = minimum j ∈ [1, 2, ··· , K] | t(j, w) ≤ t_c;
16     end
       // Generating input output pairs
17     tsDB.push(process-class(n, s, t, t_c, N));
18 end
```

ALGORITHM 2: Process workload class.

```
Input: Workload array n, class array s, time array t, timing constraint t_c and
       frame window N
Output: Input-output pairs (X^(i), y^(i)) stored in Γ
1  Initialize np.clear(), nt.clear() and Γ.clear();
   // process class to minimize voltage and frequency
      switching overhead
2  for i = 1 : N_d : |n| do
3      for l = 1 to K do
4          nt(l) = 0;
5          for j = 1 to N_d do
6              nt(l) = nt(l) + t(l, i + j − 1);
7          end
8      end
9      p = minimum l ∈ [1, ··· , K] | nt(l) ≤ N · t_c;
10     for j = i to i + N_d do
11         ns(j) = p;
12     end
13 end
   // generate input-output pairs
14 for i = 1 to |s| do
15     X = [s(i − 1)   s(i − 2)   ···   s(i − N)];
16     y = ns(i);
17     Γ.push(X, y)
18 end
```

A. Training Set Preparation

Algorithm 1 provides the pseudo-code of this step. For this purpose, a set of N_v video sequences with different resolutions are used. For each supported frequency, the hardware is set to operate at this frequency using `cpufreq-set`. Every video frame of the current video (`curr_video`) is processed at this frequency (lines 6 - 12). The performance monitoring tool `perfmon` [15] is used to record the CPU cycle count. This value is stored in the array n. The time needed for processing the frame is recorded in an array t corresponding to the selected frequency (line 10). Finally, a class (i.e., a frequency index) is assigned corresponding to every frame (lines 14 - 16). This class is the minimum frequency such that the timing constraint is satisfied. This class is stored in the array s.

The class thus obtained is processed in the `process-class` routine to minimize the overhead of the frequency switching (line 17). The pseudo-code of this routine is provided in Algorithm 2. The algorithm consists of two sections – class processing (lines 2 - 13) and generation of the input-output training pairs (lines 14 - 18). To minimize the switching overhead, the algorithm works on a window of N_d frames where no class change is allowed. A constant class is decided for every window based on the timing requirement. This is determined as follows. Every frames of a window is assigned to K classes, one at a time, to determine the total

ALGORITHM 3: Iterative parameter estimation.

Input: Training set database tsDB, number of voltage-frequency levels K, frame window N, maximum number of iterations $maxIter$ and parameter convergence criteria $pconv$
Output: Parameters of the logistic regression model (θ, γ)

1 Initialize $numIter = 1$ and $\theta = \gamma = 0$;
2 **while** $numIter \leq maxIter$ **do**
3 $pchange = \infty$;
4 **while** $pchange \geq pconv$ **do**
5 $p_1 = \theta$ and $p_2 = \gamma$;
6 **for** $u = 1$ *to* K **do**
7 **for** $v = 1$ *to* N **do**
8 $\theta_v^{(u)} = \theta_v^{(u)} + \alpha \cdot \nabla_{\theta_v^{(u)}}$;
9 Update the flip probabilities using Equation 10;
10 **end**
11 **end**
12 $pchange = \frac{|\ell(p_1,p_2) - \ell(\theta,\gamma)|}{\ell(p_1,p_2)}$;
13 **end**
14 mDB.push($\ell(\theta,\gamma)$) and pDB.push(θ,γ);
15 $numIter++$;
16 Randomly select the values of θ and γ;
17 **end**
18 Find $i \in [1, 2, \cdots, |\text{mDB}|]$ such that mDB(i) is maximum;
19 Return pDB(i);

time taken (lines 4 - 7). The time taken for the workload at a particular class is fetched from the timing array t. The total time taken by the frame window executed with a class is recorded in an array nt. The minimum value of the class that satisfies the timing requirement is selected (line 9). Finally, the class for the frames of the window is selected as this minimum class value (lines 10 - 12). This process is repeated $\lceil \frac{|n|}{N_d} \rceil$ times to determine the class of all the frames of the video sequence. Two important points to note from this section of the algorithm are: by selecting the minimum class for a workload that satisfies the timing requirement (line 9), the algorithm assigns class to workload in order to minimize energy consumption; and by selecting a constant class for a window of N_d frames, the algorithm minimizes the frequency scaling overhead. Next, the N previous workloads are stored in the vector X (line 15). These workloads are indicated as $s(i-j)\ \forall\ j \in [1, 2, \cdots, N]$, with $s(a) = 0\ \forall\ a < 1$. The current class $ns(i)$ is assigned to the output variable y. The input-output pair (X, y) is stored in the database Γ, which is returned after processing all frames.

B. Parameter Fixing

The parameter fixing step is to estimate the parameter θ of the logistic regression model by maximizing the likelihood of the probability distribution. The gradient of the log likelihood function (Equation 6) is given by

$$\nabla_{\theta_v^{(u)}} = \frac{\partial \ell(\theta)}{\partial \theta_v^{(u)}} = \sum_{i=1}^{M} \sum_{l=1}^{K} \frac{\mathcal{I}\left(\tilde{y}^{(i)} = l\right)}{\sum_{r=1}^{K} \gamma_{l,r} \cdot p_r} \sum_{r=1}^{K} \left(\gamma_{l,r} \cdot \Gamma(\theta)\right) \text{ where} \tag{8}$$

$$\Gamma(\theta) = \frac{\mathcal{I}(u = r) \sum_{j=1}^{K} g(j,i) - g(u,i)}{\left(\sum_{j=1}^{K} g(j,i)\right)^2} x_v^{(i)} g(r,i) \text{ and } g(i,j) = e^{\left(\theta^{(i)T} \cdot X^{(j)}\right)} \tag{9}$$

We use the *steepest gradient ascent* rule for the parameter (θ) update and is given by $\theta_v^{(u)} = \theta_v^{(u)} + \alpha \cdot \nabla_{\theta_v^{(u)}}$, where $0 \leq \alpha \leq 1$ is the step size (also referred to as *learning rate* in machine learning terminology). Thus, the gradient ascent technique takes a step α in the direction of the steepest increase of the function $\ell(\theta)$. The flip probabilities due to uncertainty is determined using multiplicative update [16],

$$\gamma_{j,l} = \frac{\gamma_{j,l} \sum_{i=1}^{M} \frac{\mathcal{I}(\tilde{y}^{(i)} = l}{\sum_{r=1}^{K} \gamma_{l,r} \cdot p_r} \cdot \frac{g(j,i)}{\sum_{r=1}^{K} g(r,i)}}{\sum_{l=1}^{K} \left[\gamma_{j,l} \sum_{i=1}^{M} \frac{\mathcal{I}(\tilde{y}^{(i)} = l}{\sum_{r=1}^{K} \gamma_{l,r} \cdot p_r} \cdot \frac{g(j,i)}{\sum_{r=1}^{K} g(r,i)}\right]} \tag{10}$$

Algorithm 3 shows the pseudo-code to determine the parameters of the logistic regression model. The algorithm starts from a value of θ and γ (line 1). It then updates every component of θ and γ using the iterative approach shown in lines 4 - 13. For this, a convergence criteria $pconv$ is used At every iteration (lines 4 - 13), the old values of the parameters are first stored in temporary variables p_1 and p_2. The parameters are then updated according to Equations 10 (line 8, 9). The change in the function value using these new parameters is determined (line 12). If this change is greater than $pconv$, the section is repeated using the updated value of the parameters. Once all the components of the parameters θ and γ are determined, the parameters θ and γ and the value of the log likelihood function $\ell(\theta, \gamma)$ are stored in pDB and mDB, respectively. In this work we consider $maxIter$ different random starting points for the parameters. For each of these starting points, the parameter determination procedure (lines 2 - 17) is repeated. The θ and γ corresponding to the maximum value of the log likelihood function $\ell(\theta, \gamma)$ are selected and returned.

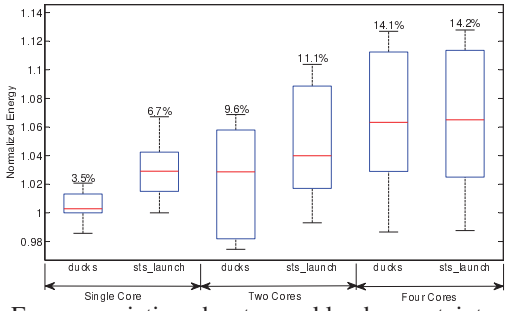


Fig. 3: Energy variation due to workload uncertainty: H.264 case study.

VII. Results

The proposed run-time approach is validated on Texas Instrument's PandaBoard featuring ARM A9 cores and Intel quad-core system running Linux. The proposed approach is compared with one representative approach from each category of related works – the reinforcement learning-based technique of [4], the prediction-based DVFS technique of [1] and the multinomial logistic regression-based technique of [9].

A. Impact of Workload Uncertainty: H.264 Case Study

To signify the impact of the workload uncertainty on the energy consumption, an experiment is conducted using two 30 sec video sequences – "ducks" and "sta_launch" [17]. The H.264 decoder application is restricted to execute on only one core using the `cpu-affinity` feature of the operating system. Further, we let all other routine applications execute freely on any cores. The videos are decoded ten times each and the energy optimization for each run is performed according to [1]. The energy values are normalized with respect to that obtained using the mean workload of the ten readings. These results are shown in Figure 3 corresponding to the label *Single Core*. The minimum energy, maximum energy, the energy median value (as red line) and the 80% energy distribution (as blue box) for the runs are shown as box plots for both the videos. Further, the percentage difference between the minimum and the maximum energy of the ten runs (referred as energy variation) is reported on top of each box. Next, the same experiment is repeated, but now allowing the H.264 decoder application to run on two and four cores of the system. The results are also plotted in Figure 3 for these two cases.

As seen from the figure, when the H.264 decoder runs on single core, there is an energy variation of 3.5% and 6.7% for the two videos respectively. It is to be noted that, even though the H.264 application executes on one core, some of the threads from other routine applications are also scheduled on this core by the operating system. Thus, there is an amount of uncertainty in the observed CPU cycle count. Hence, the voltage-frequency value obtained based on this observed CPU cycle count is not optimal, implying that there is a performance slack that enables the operating system to schedule more threads on this core. When the H.264 application is allowed to execute on two cores, the percentage variation for the two videos increases to 9.6% and 11.1%, respectively. This is because, as the application uses more cores, there are more cache conflicts due to other background threads, increasing the workload uncertainty. Finally, when all the four cores are

TABLE I: Energy consumption (Joules)

Videos	Ondemand [11]	Predictive [1]	Q-Learning [4]	MLR [9]	Proposed		
					no DO, no WN	DO, no WN	DO, WN
FFT	32.7	29.0	27.6	20.2	19.8	19.6	16.9
gsm	18.8	14.3	15.5	10.8	10.6	10.5	8.8
web_render	26.6	26.6	24.5	21.5	21.5	21.1	20.2
H.264	23.7	23.7	20.2	18.5	18.2	17.9	15.4
sobel	69.3	61.4	63.2	41.4	30.7	29.7	28.3
basicmath	82.6	82.6	60.2	42.4	40.9	40.0	37.3

used by the H.264 decoder, the energy variation increases to 14%. *It can thus be concluded that, workload uncertainty can result in as high as 14% variation in the energy results, clearly motivating our approach.*

B. Energy Savings in the Proposed Approach

Table I reports the energy savings achieved using the proposed approach in comparison with Linux's Ondemand governor [11] and three of the existing approaches. Results for the proposed approach are provided for three configurations:

No DO, No WN: In this configuration, the proposed approach performs no optimization for DVFS overhead (DO) and does not incorporate the workload uncertainty (WN). To disable the DVFS overhead optimization, we let $N_d = 1$ and to ignore the workload uncertainty, we let the class switching probabilities as 0 i.e., $\gamma_{i,j} = 0 \ \forall \ i \neq j$ and 1 for $i = j$. Line 9 of Algorithm 3 needs to be disabled for this purpose.
DO, No WN: In this configuration, the proposed approach performs optimization for DVFS overhead with $N_d = 8$, while not optimizing for the workload uncertainty. The choice of N_d is evaluated later in Section VII-C.
DO, WN: In this configuration, the proposed approach performs both these optimization simultaneously.

There are few trends to follow from this table. First, the energy optimization using workload prediction [1] is, in general, better than the linux ondemand approach (column 2 vs column 3). This is due to different optimization strategies for these two techniques. In ondemand, the frequency is increased (or decreased) based on the CPU utilization. On the other hand, predictive approaches determine the minimum frequency value based on performance. On average, the predictive approach improves energy consumption by 7% as compared to the Linux's ondemand governor. Second, Q-Learning improves energy consumption further by 15% with respect to the predictive approach (column 3 vs column 4). This energy reduction is because the Q-Learning algorithm learns the most appropriate frequency for a given workload. However, this algorithm is usually characterized by exploration phase, where different frequencies are explored to learn the appropriate value. In this phase, the algorithm may select a frequency that increases the energy consumption. Once the algorithm reaches the exploitation phase, the energy benefits become more prominent. For some application such as sobel, the Q-Learning algorithm spent most of the time in the exploration phase and therefore, the energy consumption is higher than that of the predictive approach.

Third, the logistic regression based approach of [9] achieves the best result among all the existing techniques by lowering energy consumption by an average 26% as compared to the Q-Learning approach (column 4 vs column 5). This improvement can be attributed to the fact that by offline workload characterization, the exploration phase of the Q-Learning algorithm is avoided in [9] i.e., the approach provides the optimal result from the first frame itself. The proposed approach without DVFS overhead and workload uncertainty optimization achieves 8% lower energy on average as compared to that of [9] (column 5 vs column 6). Although, the fundamental model for both the techniques are the same, the improvement in the proposed technique is due to the difference in workload prediction techniques – last frame statistics for [9] and window of past frame's CPU cycle counts for the proposed approach. When the proposed technique is applied to optimize for DVFS switching overhead (i.e., column 7), there is an improvement in the energy consumption by an average 3% (column 6 vs column 7). This improvement is due to the use of a fixed frequency for a window of N_d frames rather than changing the frequency for individual frames. Finally, the results using the proposed approach with both the features enabled i.e., the DVFS overhead and workload uncertainty, is reported in column 8. The improvement with respect to DVFS optimization alone (i.e., column 7 vs column 8) is 11%, while that with respect to no features enabled (column 6 vs column 8) is 13%. These results indicate that the proposed technique minimizes the workload uncertainty effectively. Finally, the improvement using the proposed technique with respect to the existing logistic regression technique of [9] is on average 20%. To summarize, *there are three aspects that contribute to the 20% improvement in the proposed approach – workload uncertainty (achieving 11% improvement), frame history-based workload prediction (achieving 8% improvement) and DVFS overhead optimization (achieving 3% improvement).* It is to be noted that, the user is provided with the flexibility to enable or disable any of the additional features of the proposed approach.

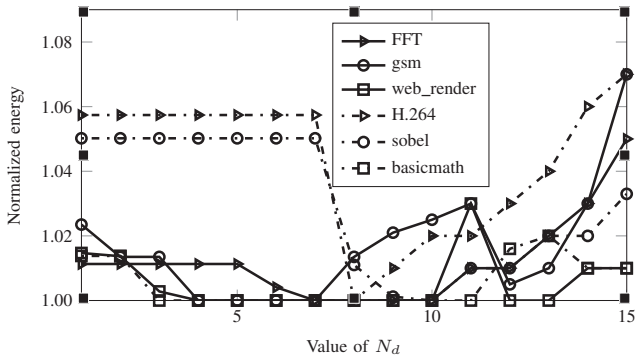


Fig. 4: Impact of the DVFS window N_d.

C. Impact of Design Parameters

There are three design choices for the proposed logistic regression model – the DVFS window N_d, the prediction frame window N and the number of training samples M.

1) DVFS Window: Figure 4 plots the variation of the normalized energy obtained by varying the DVFS window N_d for the same six applications. $N_d = 1$ implies no DVFS optimization. The energy values for each video are normalized with respect to the minimum energy values obtained for the corresponding video with N_d varying from 1 to 15. The general trend that can be seen from this figure is that, as the DVFS window size increases, there is first a decrease in the energy consumption and then the energy consumption increases. This decrease is due to the reduction of the frequency switching overhead. However, as the size of the window increases further, the approach enforces the same voltage-frequency value for a large number of frames, reducing the opportunity of energy reduction using DVFS. The minimum energy point differs for different applications. We chose $N_d = 8$ as this gives the best result for all applications.

2) Prediction Frame Window: The prediction frame window N is the window of past frames that are used to predict the next frame. It is the size of the input vector X in Section V. Figure 5 plots the impact of varying the size of N on the workload misprediction for six applications. Workload misprediction is defined as the difference between the actual workload and the predicted workload. Thus, lower the workload misprediction, the more accurate the estimation (and hence better). The above figure plots the root mean square (rms) of the workload misprediction as the value of N is increased. The rms values are normalized with respect to the rms value obtained using $N = 1$ i.e., predicting the next frame using the last frame only. As can be seen, for some applications (FFT, gsm and web_render), the rms of the workload misprediction at $N = 10$ is very close (average 6%) to the rms value obtained with $N = 1$. This is because,

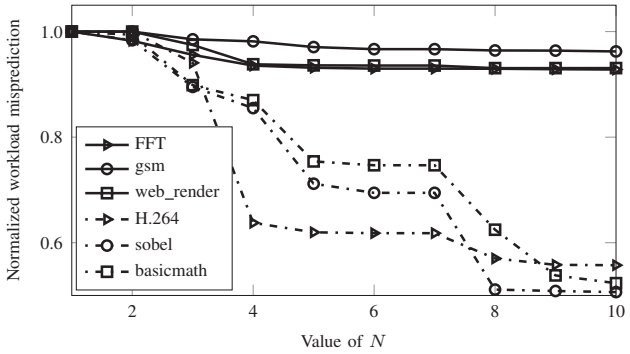


Fig. 5: Impact of the prediction frame window N.

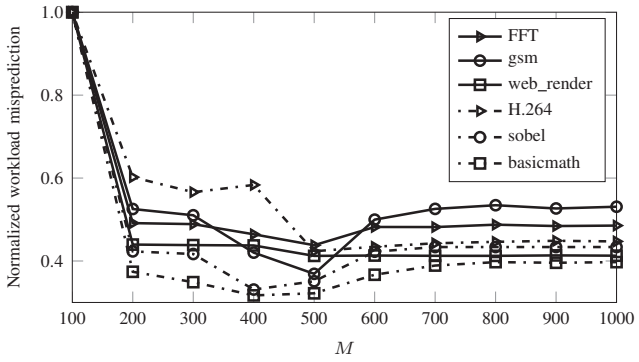


Fig. 6: Impact of the number of training samples M.

these applications are characterized with workloads that vary little with time (static workload), and therefore there is no significant improvement in considering a window of past frames as compared to using the last frame statistics only. This justifies our previous observation that the proposed technique is only 1% better than that of the existing technique of [9] for these applications. On the other hand for for applications with dynamic workload, the average decrease in the rms of the workload misprediction is 47% for $N = 10$. This implies that for these applications, it is essential to consider a window of past frame statistics in order to predict the future frames. Although not shown in this figure, beyond $N = 10$, there is very little improvement in the misprediction (less than 0.01%), suggesting that 10 previous frames are sufficient to accurately predict the next frame for all applications.

3) Training Samples: The number of training samples (or X, y pairs) determines the quality of workload class prediction using the logistic regression model. Figure 6 plots the root mean square of the workload prediction error as the size of M is varied from 100 to 1000. The workload misprediction values obtained using the proposed approach are normalized with respect to the misprediction for M = 100. As can be seen, the workload misprediction decreases with an increase in the number of training samples. This is expected because, as the number of training samples increases, there is higher probability that the model encounters different workload scenarios that help the model to learn better and predict the application workload accurately (decrease in workload misprediction). However, beyond 600 input-output pairs, there are no change in the misprediction values. Further, a higher M implies higher model development time (refer to Section VII-D).

D. Execution Time of the Proposed Approach

The execution time of the approach is calculated as follows. The training set preparation time is dependent on the size of the training videos used and the number of frequencies supported on the hardware. The wall clock time recorded for the training set preparation was 20.5 minutes. Figure 7 plots the execution time of the parameter fitting step of the proposed algorithm for different values of the frame window N and the sample size M. As can be seen, with an increase in the number of samples, the execution time increases. Finally, the run-time overhead of the multinomial logistic regression based approach is computed as follows: the class prediction for the next workload frame takes an average 0.7ms; fetching the voltage-frequency value corresponding to each class takes an average 0.5μs; and the time taken by the operating system to set the voltage-frequency value on the cores (i.e., the time for the `cpufreq-set` command) is on average 0.5ms. These numbers are collected based on the average of 1000 runs. Thus, the overall run-time overhead for DVFS is 1.2ms.

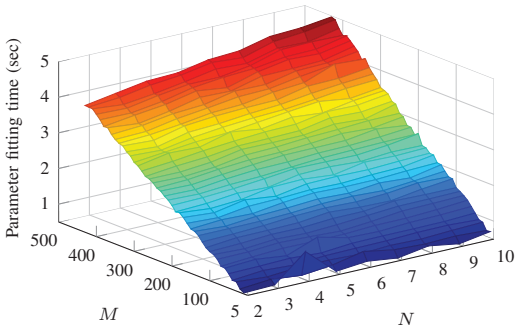


Fig. 7: Execution time for different N and M.

VIII. Conclusions

In this paper we proposed an adaptive approach for energy minimization of multimedia applications on a multicore system. Fundamental to this approach is a multinomial logistic regression based workload classification technique that classifies the incoming video workloads in the presence of uncertainty into a discrete set of classes. Every class is associated with a voltage-frequency value pre-determined using an appropriate training set and results in minimum energy consumption. The proposed approach is engineered on a multicore system running Linux. Experiments conducted with multimedia applications demonstrate that the proposed approach minimizes energy consumption by an average 20% with no performance degradation.

Acknowledgment

This work was supported in parts by the EPSRC Grant EP/L000563/1 and the PRiME Programme Grant EP/K034448/1 (www.prime-project.org).

References

[1] H. Jung and M. Pedram, "Continuous Frequency Adjustment Technique Based on Dynamic Workload Prediction," in *VLSI Design*, 2008.
[2] G. Dhiman and T. S. Rosing, "Dynamic Voltage Frequency Scaling for Multi-tasking Systems Using Online Learning," in *ISLPED*, 2007.
[3] H. Shen, J. Lu, and Q. Qiu, "Learning based DVFS for Simultaneous Temperature, Performance and Energy Management," in *ISQED*, 2012.
[4] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving Autonomous Power Management Using Reinforcement Learning," *ACM TODAES*, 2013.
[5] R. Ye and Q. Xu, "Learning-Based Power Management for Multicore Processors via Idle Period Manipulation," *IEEE TCAD*, 2014.
[6] G. Dhiman, V. Kontorinis, D. Tullsen, T. Rosing, E. Saxe, and J. Chew, "Dynamic Workload Characterization for Power Efficient Scheduling on CMP Systems," in *ISLPED*, 2010.
[7] H. Jung and M. Pedram, "Supervised Learning Based Power Management for Multicore Processors," *IEEE TCAD*, 2010.
[8] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps," in *MICRO*, 2011.
[9] R. Cochran, C. Hankendi, A. Coskun, and S. Reda, "Identifying the Optimal Energy-Efficient Operating Points of Parallel Workloads," in *ICCAD*, 2011.
[10] P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini, "Workload and User Experience-aware Dynamic Reliability Management in Multicore Processors," in *DAC*, 2013.
[11] V. Pallipadi and A. Starikovskiy, "The Ondemand Governor," in *Proceedings of the Linux Symposium*, 2006.
[12] L. Benini, A. Bogliolo, and G. De Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *TVLSI*, 2000.
[13] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, *Machine Learning, Neural and Statistical Classification*. New York: Ellis Horwood, 1994.
[14] J. A. Anderson and S. C. Richardson, "Logistic Discrimination and Bias Correction in Maximum Likelihood Estimation," *Technometrics*, 1979.
[15] "Perfmon: Linux Profiling with Performance Counters," *Downloadable software with documentation, http://www. hpl. hp. com/research/linux/perfmon*, 2012.
[16] J. Bootkrajang and A. Kabn, "Label-Noise Robust Logistic Regression and Its Applications," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, 2012.
[17] Derf. (2014) Test Media. [Online]. Available: http://media.xiph.org/video