Program Phase and Runtime Distribution-Aware Online DVFS for Combined Vdd/Vbb Scaling

Jungsoo Kim*, Sungjoo Yoo†, and Chong-Min Kyung*

*Dept. of EECS at KAIST
jskim@vslab.kaist.ac.kr, kyung@ee.kaist.ac.kr

†Dept. of EE at POSTECH,
sungjoo.yoo@postech.ac.kr

***Abstract*—Complex software programs are mostly characterized by phase behavior and runtime distributions. Due to the dynamism of the two characteristics, it is not efficient to make workload predictions during design-time. In our work, we present a novel online DVFS method that exploits both phase behavior and runtime distribution during runtime in combined Vdd/Vbb scaling. The presented method performs a bi-modal analysis of runtime distribution, and then a runtime distribution-aware workload prediction based on the analysis. In order to minimize the runtime overhead of the sophisticated workload prediction method, it performs table lookups to the pre-characterized data during runtime without compromising the quality of energy reduction. It also offers a new concept of program phase suitable for DVFS. Experiments show the effectiveness of the presented method in the case of H.264 decoder with two sets of long-term scenarios consisting of total 4655 frames. It offers 6.6% $\sim$ 33.5% reduction in energy consumption compared with existing offline and online solutions.**

I. Introduction

Dynamic voltage and frequency scaling (DVFS) is one of the most effective low power design methods. Due to the increasing leakage power consumption, DVFS now controls both supply voltage (Vdd) and body bias (Vbb) dynamically to minimize the total power consumption [1]. DVFS sets the performance level of CPU to the ratio of predicted remaining workload to the given deadline. Thus, the accuracy of remaining workload prediction (in short, workload prediction) plays a crucial role in obtaining minimum energy consumption.

The workload prediction is to predict future workload mostly based on recent history. For instance, the average of recent workload can be a workload prediction. However, in reality, due to the complex behavior of software program (e.g., data dependent iteration counts of nested loops) and architectural factors (e.g., cache miss, DDR memory page miss, etc.), such a naive prediction may not work efficiently. Fig. 1 (a) illustrates a profile of per-frame workload in H.264 decoder (an excerpt from the movie "*Lord of the Rings*"). The X-axis and the left-hand side Y-axis represent frame index and per-frame workload, respectively. The figure shows that the profile has two different scales of behavior in both macroscopic and microscopic ways. First, it has a macroscopic time-varying behavior, i.e., phase behavior. There are time durations whose workload characteristics (e.g., mean, standard deviation, max value, etc.) are distinctly different from other time durations. We call a time duration with a distinct workload characteristic a *phase* (we will give a formal definition later in this paper). Fig. 1 (a) shows 10 phases (see the right-hand side Y-axis for phase indexes). Note that the phase index does not correspond to the required performance level of the corresponding phase in this example. The example of Fig. 1 also shows the microscopic behavior, the runtime distribution. Fig. 1 (b) gives the runtime distributions of three representative phases to illustrate that there can be a wide runtime distribution even within a phase, and the phase itself is characterized by the runtime distribution.

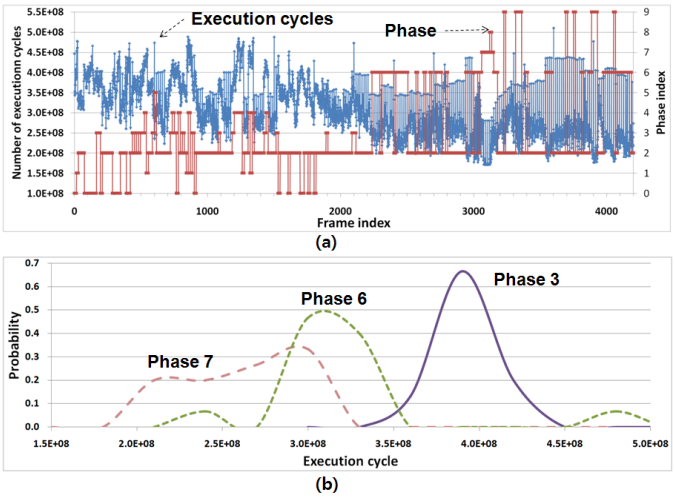

Fig. 1. Example of phase behavior and runtime distribution: (a) phase behavior in the per-frame workload of H.264 decoder and (b) runtime distributions (PDFs)

Our observation on the runtime characteristics of software programs suggests that, as shown in Fig. 1, the program workload has two characteristics: phase behavior and (multi-modal) runtime distribution (even within a phase). Especially, a phase can have a multi-modal runtime distribution with more than one salient peaks as phases 6 and 7 in Fig. 1 (b) show[1]. Considering the worst-case execution time has a significant impact on the efficiency of DVFS for real-time systems, such a multi-modal distribution needs to be carefully analyzed and exploited in order to obtain an accurate prediction of remaining workload.

In our work, we aim at an online DVFS method that exploits both phase behavior and multi-modal runtime distribution in

[1]Note that the single mode distribution can be considered to belong to the multi-modal distribution

978-3-9810801-5-5/DATE09 © 2009 EDAA

order to make accurate workload predictions for dynamic Vdd/Vbb scaling. We address the online DVFS problem in two ways: intra-phase workload prediction and phase detection. The intra-phase workload prediction is to predict workloads based on the runtime distribution of the current phase. The phase detection is to identify to which phase the current instant belongs. To the best of the authors' knowledge, our work is the first approach of online DVFS for real-time systems which exploits both phase behavior and runtime distribution in combined Vdd/Vbb scaling.

This paper is organized as follows. Section II reviews existing work. Section III presents an overall flow. Section IV explains a multi-modal (i.e., bi-modal) analysis of runtime distribution. Section V gives the details of workload prediction. Section VI presents the phase detection method. Section VII reports experimental results, followed by the conclusion in Section VIII.

II. Related Work

There have been presented a lot of research works on the workload prediction for online DVFS, e.g., (weighted) average of N recent workloads [2]. Recently, a control theory-based prediction method is presented [3]. The above studies are effective in the case of simple workload characteristics without phase behavior or wide runtime distributions.

Phase detection has been one of hot research issues since it will allow for new opportunities of performance optimization, e.g., dynamic adaptations of cache architecture [4] [5]. Phase detection is also applied to DVFS in [6] [7]. In this work, the per-phase runtime characteristic is modeled with a vector of execution cycles of basic blocks. A new phase is detected when two vectors are significantly different, e.g., when there is a large Hamming distance between the two vectors. The key problem here is to identify a subset of basic blocks that represent phase behavior. Exploring all the combinations of basic blocks will be prohibitively expensive in the case of current and future complex software applications with a large number of basic blocks. In this paper, we present a practical method of phase detection, suitable for DVFS purpose, which is based on the vectors of predicted workloads for coarse grain code sections as explained in Section VI. In addition, compared with existing phase-based DVFS methods, the presented method exploits runtime distribution within a phase to better predict the remaining workload.

Runtime distribution has been actively exploited mostly in offline DVFS methods. Control flow-dependent time slack is exploited by predicting the remaining workload on a path basis in most of intra-task DVFS methods: the worst-case execution path [8] [9], average-case execution path [10], and virtual execution path [11]. Recently, analytical approaches have been presented to address all the sources of runtime distribution: data dependency (e.g., number of loop iterations), and architecture (e.g., cache misses) as well as control flow [12] [13].

Existing offline runtime distribution-aware DVFS methods, if applied to online DVFS, would suffer from two limitations. First, they lack in utilizing phase behavior. In these methods, a single runtime distribution is obtained by running all the test benches over possibly numerous phases. Thus, phase-specific workload information is lost, which may lead to inefficiency in lowering energy consumption for software programs with noticeable phase behavior. Second, if they are applied to online DVFS without modifications, they will incur prohibitively high runtime overhead due to its computation complexity (e.g., up to 2.5 times of entire program runtime in solving differential equations numerically [13] as explained in Section VII). In order to overcome these limitations, we present a low overhead online version of originally offline runtime distribution-aware DVFS method.

Algorithm 1: Overall flow

```
1: if (Time % PHASE_UNIT==0) then
2:    for from N_leaf to N_root do
3:       Bi-modal analysis of runtime distribution
4:       Workload prediction
5:    end for
6:    Check whether a new phase starts
7: end if
```

III. Overall Flow for Online DVFS

Algorithm 1 shows the overall flow of the proposed method. Our work focuses on intra-task DVFS where the performance level is set at each performance setting point (PSP) inserted into the software code by designers or automatically.

We perform workload prediction and phase detection periodically (e.g., on a granularity of PHASE_UNIT cycle period in line 1 of Algorithm 1. Note that a phase can consist of multiple consecutive periods (e.g., each period with PHASE_UNIT cycles). On every period, PSPs are traversed from the end of program (N_{leaf}) to the beginning of program (N_{root}) for workload prediction (lines $2 \sim 5$). At each PSP, we perform a bi-modal analysis of runtime distribution (line 3) and predict the remaining workload (line 4). We approximate a multi-modal distribution with a bi-modal one, since, in most cases, the number of modes is less than or equal to two. Thus, our approximation does not incur a significant inefficiency in energy reduction as Section VII shows. The phase detection check is performed (line 6) utilizing the predicted workloads. The phase detection is to identify to which phase the current period belongs. A new phase is detected when there is a large difference (in terms of Hamming distance of PSP vectors, to be explained in Section VI) between the predicted workload of current phase and that of current period.

Fig. 2 illustrates the workload prediction based on the bi-modal analysis. Fig. 2 (a) shows two program regions, n_i and n_{i+1} (a program region is a code section starting with a PSP and finishing with another PSP). Fig. 2 (b) shows the PDF (probability distribution function) of runtime distribution for each program region and the key steps of workload prediction for the program region n_i in this case.

Given the runtime distribution of a phase, in order to predict the energy optimal remaining workload for combined

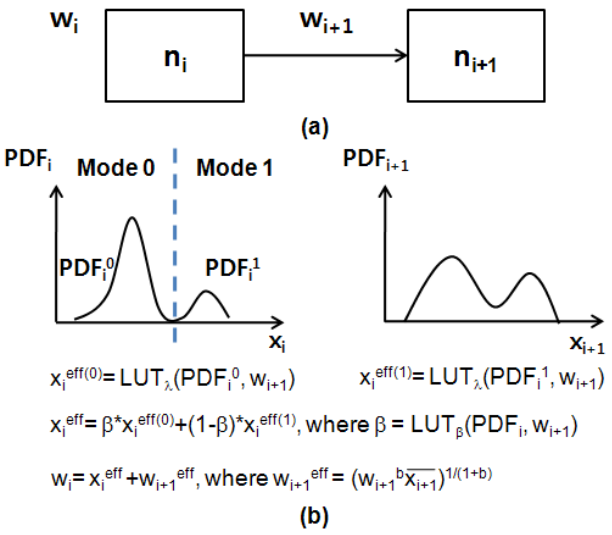

Fig. 2. Workload prediction: (a) two program regions and (b) workload prediction with the bi-modal analysis

Vdd/Vbb scaling, we apply the solution presented in [13] during runtime. However, a direct execution of the solution during the software program run on the target CPU will cause prohibitively high overhead of runtime and energy consumption as shown in Section VII. For the online purpose, we need a lightweight, yet accurate solution to the workload prediction. In this paper, we propose an approach that gives a low runtime overhead by exploiting the pre-characterized data. To do that, we first modularize the remaining workload from program region n_i to the end of program into two parts: one (called the effective workload of program region n_i denoted by x_i^{eff}) mostly determined by the PDF of program region n_i and the other (called the effective workload of remaining program regions except n_i denoted by w_{i+1}^{eff}) mostly determined by the PDFs of remaining program regions. At the PSP for program region n_i, we predict the remaining workload w_i as the sum of x_i^{eff} and w_{i+1}^{eff}. x_i^{eff} is calculated by table lookups to the pre-characterized data (in the form of two types of look-up table) with the runtime distributions (obtained during runtime) as the input to the table lookups. w_{i+1}^{eff} is obtained analytically (Section V). Fig. 2 (b) also illustrates how each of x_i^{eff} and w_{i+1}^{eff} is calculated. In order to obtain x_i^{eff}, the bi-modal analysis decomposes the PDF of n_i, PDF_i into two parts called modes 0 and 1 as Fig. 2 (b) shows. For each mode, we calculate $x_i^{eff(0)}$ and $x_i^{eff(1)}$ called the effective workload of mode by using the lookup table called LUT_λ (details will be given in Section V). Then, we obtain the effective workload of program region n_i by the weighted sum of the two effective workloads of mode by looking up the other table LUT_β for the parameter β (Section IV). The effective workload of remaining program regions other than n_i, i.e., w_{i+1}^{eff}, is calculated in a simple manner as Fig. 2 (b) shows (Section V). Finally, the summation of the two effective workloads gives the predicted remaining workload of program region n_i, w_i as the lowest equation in Fig. 2 (b) shows.

Algorithm 2: Mode decomposition

1: find two non-continuous points (x_0, p_0) and (x_1, p_1) whose probability values, p_0 and p_1 are the two highest probabilities in the original PDF
2: **if** there are no such two non-continuous points, **then**
3: the entire PDF is considered to be a single mode
4: return the original PDF
5: **else**
6: find the saddle point (x_s, p_s) between the two points, which gives the minimum probability
7: **if** there are more than on saddle points **then**
8: the median is selected as the saddle point
9: **end if**
10: $PDF^0 = \{(x,p)|x \leq x_s, (x,p) \in PDF\}$
11: $PDF^1 = \{(x,p)|x > x_s, (x,p) \in PDF\}$
12: return PDF^0 and PDF^1
13: **end if**

At each PSP, the performance level is set to the ratio of predicted workload to the remaining time-to-deadline or to a level that satisfies the given deadline constraint depending on the result of real-time constraint check as in [13]. Note that the performance setting implies Vdd/Vbb setting since there is a one-to-one correspondence between a performance level and a pair of Vdd/Vbb settings that give the minimum energy consumption [1].

IV. Bi-modal Analysis

We calculate the effective workload of program region n_i, i.e., x_i^{eff}, in three steps: mode decomposition, workload prediction for each mode, and mode recomposition to obtain the effective workload of the program region.

A. Mode Decomposition

Mode decomposition is to decompose the original runtime distribution into two modes, i.e., two separated distributions each of which has a salient peak. Algorithm 2 explains how to decompose the original runtime distribution into two modes. As Algorithm 2 shows, the original PDF is decomposed into two sub-PDFs, PDF^0 and PDF^1 at a saddle point where the probability is the minimum. In the case that there is only one mode (lines 2 $\sim$ 4 in Algorithm 2), the original PDF is returned.

B. Modes Recomposition

In the step of workload prediction (Section V), the effective workload for each of the two modes ($x_i^{eff(0)}$ and $x_i^{eff(1)}$) is obtained. Then, the effective workload of program region n_i is calculated as a weighted sum of the two values as follows.

$$x_i^{eff} = \beta x_i^{eff(0)} + (1-\beta) x_i^{eff(1)} \quad (1)$$

Parameter β determines the relative importance of two modes. In our work, we calculate the parameter by a table lookup of pre-characterized data with the runtime distributions of program regions as the input of table lookup. In the following, we explain how to build the lookup table for the accurate parameter calculation.

The two modes will have different impacts on the final predicted remaining workload depending on three factors as follows.

- Factor 1: Ratio between the relative probabilities of the two modes in the original distribution
- Factor 2: Ratio between the execution cycles of the two modes
- Factor 3: Ratio between the execution cycle of program region n_i and the remaining workload after the program region n_i

For instance (Factor 1), if mode 0 has a significant portion, i.e., high probability, parameter β will have a high value approaching '1'. As another example, if mode 1 has much higher execution cycles than mode 0, i.e., if WCEC (worst-case execution cycle) is much higher than AEC (average execution cycle), then mode 1 has more impact than mode 0 since DVFS tends to set a high frequency (i.e., high effective workload of program region n_i) due to the large WCEC in order to meet the given deadline constraint. In such a case, parameter β becomes a small value to reduce the effect of mode 0 and to increase that of mode 1.

We prepare a pre-characterized table for parameter β, LUT_β with the above three factors as the index. To be specific, Factor 1 is represented by the cumulative probability of mode 0, P_0 (since $P_0 + P_1 = 1$). Factor 2 is represented by the ratio of $x_i^{eff(1)}$ to $x_i^{eff(0)}$ since each of them represents the execution cycle information of each mode. Factor 3 is represented by the ratio of $(x_i^{eff(0)} + x_i^{eff(1)})/w_{i+1}^{eff}$.[2]

As $x_i^{eff(1)}/x_i^{eff(0)}$ increases (e.g., WCEC $\gg$ AEC) or $x_i^{eff(1)}/ w_{i+1}^{eff}$ increases, parameter β decreases since mode 1 comes to have more impact on the energy optimal remaining workload than mode 0. Regarding $x_i^{eff(1)}/w_{i+1}^{eff}$, as a simple case, if w_{i+1}^{eff} approaches 0, then the program region n_i dominates the remaining workload. Thus, the energy optimal remaining workload of program region n_i will approach the worst-case execution cycle of program region n_i. Thus, mode 1 (higher portion of PDF) dominates the remaining workload, which requires parameter β (1-β) to decrease (increase) in Eqn. (1).

As a summary, when the PDFs of program regions are available during runtime by performance monitoring the four variables, P_0, $x_i^{eff(1)}$, $x_i^{eff(0)}$, and w_{i+1}^{eff} are calculated as explained in Section V. Then, the table LUT_β is looked up for the parameter β, which is used in the calculation of Eqn. (1).

V. Predicting Remaining Workload for a Decomposed Mode

In this section, we explain how to calculate the effective workload, i.e., x_i^{eff}, assuming a single (decomposed) mode. Our approach is based on an analytical formulation utilizing an analytical energy function. Combined Vdd/Vbb scaling does not have the quadratic relationship between energy consumption per cycle and frequency that the Vdd-only scaling has.

[2]In our implementation, we use the ratio of $x_i^{eff(1)}/w_{i+1}^{eff}$ since, given a ratio of $x_i^{eff(1)}/x_i^{eff(0)}$, $(x_i^{eff(0)} + x_i^{eff(1)})/w_{i+1}^{eff}$ and $x_i^{eff(1)}/w_{i+1}^{eff}$ represent the same information.

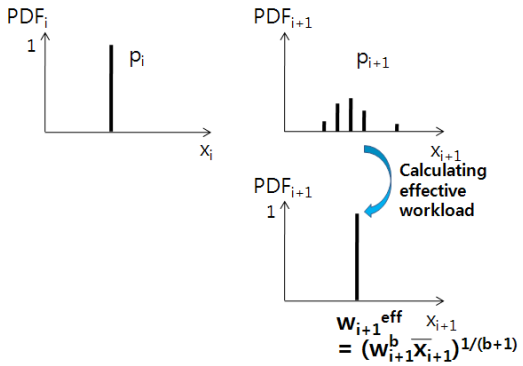

Fig. 3. Effective remaining workload, w_{i+1}^{eff}

Thus, we approximate the energy consumption by fitting the golden energy model (measurement data or estimation result) as follows.

$$E_{cycle} = af^b + c \quad (2)$$

where E_{cycle} is the energy consumption per cycle, and parameters a, b and c are fitting parameters[3].

A. Effective Workload of Program Region

Fig. 3 illustrates the PDFs of two program regions, n_i and n_{i+1} as in Fig. 2 (a). For simplicity, we assume a unit function for PDF_i. We will generalize the case (i.e., utilize a general form for PDF_i) later in this section. Given the PDFs (PDF_i and PDF_{i+1} in Fig. 3), the average energy consumption of two program regions, i.e., $\overline{E}(w_i)$, and the energy optimal remaining workload of program region n_i, i.e., w_i, are calculated using the energy model in Eqn. (2) as follows[4].

$$\begin{aligned} E(x_i, x_{i+1}, w_i) &= (af_i^b x_i + cx_i) + (af_{i+1}^b x_{i+1} + cx_{i+1}) \\ \overline{E}(w_i) &= \int\int E(x_i, x_{i+1}, w_i) p_i p_{i+1} dx_i dx_{i+1} \\ &= aw_i^b x_i + \frac{aw_{i+1}^b \overline{x_{i+1}}}{(1 - x_i/w_i)^b} + cx_i + c\overline{x_{i+1}} \\ \frac{\partial \overline{E}(w_i)}{\partial w_i} = 0 &\Longrightarrow x_i + w_{i+1}^b \overline{x_{i+1}} \cdot [\frac{-x_i}{(w_i - x_i)^{b+1}}] = 0 \\ w_i &= x_i + (w_{i+1}^b \overline{x_{i+1}})^{1/(b+1)} = x_i + w_{i+1}^{eff} \quad (3) \end{aligned}$$

As shown in Eqn. (3), the predicted remaining workload of program region n_i, w_i consists of the workload of program region, i.e., x_i, and the second term, i.e., $(w_{i+1}^b \overline{x_{i+1}})^{1/(b+1)}$. The second term represents the portion of remaining workload after program region n_i. We call it the *effective remaining workload of* n_{i+1}, w_{i+1}^{eff}. Fig. 3 illustrates that w_{i+1}^{eff} represents the PDFs of remaining program regions after n_i.

Assume the general case where program region n_i also has a wide PDF. In this case, we need to apply the numerical solution in [13] to obtain the energy optimal remaining workload. However, if such an analysis is performed during runtime, it

[3]The energy model can be derived analytically and numerically [14].

[4]For the sake of explanation, we assume the unit delay as the remaining time to the given deadline at n_i, i.e., $T_i = 1$. Thus, we set the frequency of program region n_i to $f_i = w_i/T_i = w_i$. In Eqn. (3), the frequency of program region n_{i+1}, f_{i+1} becomes $w_{i+1}/(T_i - x_i/f_i) = w_{i+1}/(1 - x_i/w_i)$. More detailed derivation can be found in [12]

will cause prohibitively large runtime overhead. Thus, being inspired by Eqn. (3), we model the solution, w_i as follows.

$$w_i = x_i^{eff} + w_{i+1}^{eff} \quad (4)$$

where x_i^{eff} is the *effective workload of program region* n_i. Note that w_{i+1}^{eff} is obtained as Eqn. (3) shows. In the case that the software program has cascaded program regions and conditional branches, we calculate the effective remaining workload of program region in a similar manner to [13].

We calculate x_i^{eff} (for each of two modes in Section IV) by exploiting the pre-characterization of solutions. To do that, we represent x_i^{eff} as follows.

$$x_i^{eff} = \overline{x_i} \cdot (1 + \lambda) \quad (5)$$

Then, we prepare a lookup table, LUT_λ for the residue λ during design-time and perform table lookups during runtime to obtain λ. We derived the indexes of LUT_λ as follows: σ_i/μ_i, γ_i, w_{i+1}^{eff}/μ_i, where μ_i, σ_i, and γ_i represent the mean, standard deviation, and skewness of PDF_i, respectively. The rationale of choosing the three indexes is as follows. First, the residue λ depends on μ_i, w_{i+1}^{eff}, and PDF_i as Appendix explains. The PDF of a mode is modeled as a skewed normal distribution since the decomposed mode usually does not have a nice normal distribution though there is mostly one salient peak per mode. Thus, there can be some level of skewness (γ_i) in the PDF of decomposed mode. The dependence of residue λ on μ_i, w_{i+1}^{eff}, and the skewed normal approximation of PDF (σ_i,γ_i) gives the above three indexes of LUT_λ.

VI. Phase Detection

A phase needs to be characterized by a salient difference in program behavior, especially, in terms of execution cycles. Conventionally, the phase is represented by a vector of execution cycles of basic blocks [4] [5]. As mentioned in Section II, a direct application of the existing phase definition may not be effective in DVFS. In our work, we first define a new vector, called PSP vector, which consists of predicted remaining workloads of program regions. Then, we detect a new phase when the Hamming distance between the representative PSP vector of current phase and that of current period becomes greater than a threshold (set to 10% in our experiments). The rationale is that the predicted remaining workload of a program region represents the entire runtime distributions of remaining program regions. Thus, it can be a representative of future behavior.

The representative PSP vector of a phase is calculated as the median vector of all the PSP vectors of periods belonging to the phase. After the phase detection, in order to utilize the phase-level repetitive behavior, we check to see if there is any previous phase similar to the newly detected one by comparing the PSP vector of the current period and the representative PSP vectors of ever existed phases. If so, we reuse the runtime distribution of the matched previous phase as that of the new phase. If there is no previous phase similar to the new one, then a new phase starts by maintaining a new set of runtime distribution information until another new phase is detected.

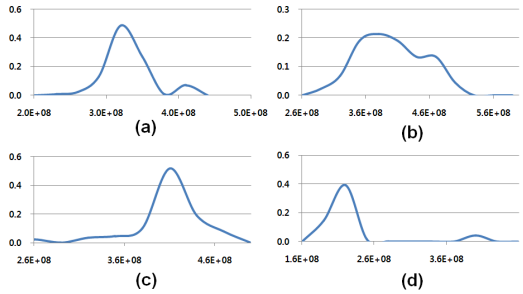

Fig. 4. Runtime distribution of test pictures: (a) foreman, (b) football, (c) stefan, and (d) akiyo

VII. Experimental Results

A. Experimental Setup

We use a real software program, H.264 decoder (QCIF, 10fps) in the experiments. In order to investigate the effects of phase and runtime distribution, we used total 4655 frames of pictures. The examples consist of two sets as follows.

- Set 1: A sequence of conventional test pictures: foreman 127 frames → football 89 frames → Stefan 89 frames → akiyo 150 frames (total 455 frames). Fig. 4 shows the runtime distributions (PDFs) of test pictures obtained from cycle-accurate simulations.

- Set 2: Four movie clips from "Lord of the Rings" (total 4200 frames). The per-frame runtime is shown in Fig. 1 (a).

We use the processor model with combined Vdd/Vbb scaling in [13]. We use 11 frequency levels up to 6.0GHz with 0.5GHz step size[5]. We run cycle-accurate simulation with a commercial tool, ARM SoCDesigner in order to obtain the PDFs for all the program regions in the software program.

B. Results

We compared the energy consumption of five methods: two offline methods and three online ones. As the offline methods, we use an average execution-cycle based method (AEC) and a runtime distribution-aware method (DIST), both from [13]. For the online methods, we use a control theory-based method (CON) [3], phase-aware average execution cycle-based one (P-AEC), and the presented one (OURS). Regarding the control theory-based method, we used the coefficients reported in [3] as the initial ones and made a further exploration of coefficients to obtain the best results. The phase-aware average execution cycle-based method, which we also present in this paper, is to exploit only the phase behavior while predicting the remaining workload based on the average remaining execution cycle obtained from the history. In this case, the phase detection is performed based on the Hamming distance in the vectors of average execution cycles of program regions.

Tables I (a) and (b) show the energy consumptions for Sets 1 and 2, respectively. All the energy consumption data

[5]We set the maximum frequency at 6GHz due to the tight deadline constraint of H.264 decoder, 10fps. We will be able to use lower maximum frequency if the deadline constraint is relaxed. The presented method still works in both cases.

(a)

Oracle DVFS (nJ)	Offline solution		Online solution		
	AEC	DIST	CON	P-AEC	OURS
6.37E+8	1.34	1.26	1.33	1.32	1.14

(b)

	Oracle DVFS (nJ)	Offline solution		Online solution		
		AEC	DIST	CON	P-AEC	OURS
Clip 1	6.96E+8	1.43	1.41	1.46	1.20	1.13
Clip 2	6.59E+8	1.48	1.42	1.45	1.23	1.16
Clip 3	4.96E+8	1.23	1.12	1.26	1.14	1.05
Clip 4	4.80E+8	1.68	1.55	1.69	1.30	1.12
All	5.78E+8	1.53	1.46	1.52	1.22	1.11

TABLE I
ENERGY CONSUMPTION FOR (A) SET 1 (B) SET 2

are normalized into the energy consumption of Oracle-DVFS which predicts the remaining workload perfectly.

Analysis

Table I shows that the presented method gives 6.6% $\sim$ 33.5% (6.6% $\sim$ 28.2%, compared with only DIST) further reductions in energy consumption than the existing offline methods. Such a significant improvement results from the fact that the presented method exploits phase behavior while still applying the runtime distribution-aware workload prediction. Compared with the control theory-based method, the presented method gives 13.9% $\sim$ 33.9% lower energy consumption. The main reason of such a large improvement is that H.264 pictures have (random-like) fast changing local runtime variations as well as phase behavior as Fig. 1 shows. The control theory-based method does not track well the fast changing local runtime variations. The phase-aware average execution cycle-based method gives better results than the control theory-based one since it has an effect of averaging out the local random workload variations. The presented method even offers 5.6% $\sim$ 14.0% further improvements over P-AEC. The improvements are obtained from two differences: (1) workload prediction based on runtime distribution (OURS) or average (P-AEC), and (2) phase detection based on runtime distribution (OURS) or average (P-AEC).

Runtime Overhead

We measured the runtime overhead of the proposed online workload prediction method when running H.264 decoder with "Lord of the Rings" (4200 frames) on ARM926 processor. We set PHASE_UNIT and the number of program regions as 15 frames and eight, respectively. Under the condition, the runtime overhead ranges from 913,761 $\sim$ 1,003,178 clock cycles, which corresponds to 0.021% of the total execution cycles. Compared with the runtime overhead of the presented online method, that of the design-time solution [13], if the design-time solution is applied during runtime without modification, amounts to about 12 billion clock cycles under the same condition as above. It is 12,000 times bigger than that of the presented online method. Such a high overhead is unacceptable since the runtime overhead alone takes 2.5 times longer runtime than that of H.264 decoder run.

Memory Overhead of LUTs

The presented method requires two types of LUTs: LUT_β and LUT_λ . The LUTs require memory space. The memory overhead largely depends on the number of steps (scales) in the indexes of the tables. As the numbers of steps increase, more accurate workload prediction will be achieved with a higher memory area overhead. In our implementation, the total area overhead of LUTs is 20kB by adjusting the step sizes and by compressing the contents of LUTs while exploiting the value locality in the tables.

VIII. CONCLUSION

In this paper, we presented a novel online DVFS method that utilizes both phase behavior and runtime distribution to give accurate workload predictions thereby lower energy consumption in combined Vdd/Vbb scaling. It performs a bi-modal analysis to practically account for the multi-modal characteristics of runtime distribution. The runtime distribution-aware workload prediction is executed while exploiting the pre-characterized data in order to minimize the runtime overhead of online method. For the phase detection for DVFS, a new concept of phase is presented which is based on the runtime distribution of program regions. Experimental results show that the presented method offers 6.6% $\sim$ 33.9% further energy savings compared with existing offline and online methods.

REFERENCES

[1] S. M. Martin, *et al.*, "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads," Proc. ICCAD, 2002.

[2] K. Govil, *et al.*, "Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU," Proc MOBICOM, 1995.

[3] Y. Gu, *et al.*, "Control Theory-based DVS for Interactive 3D Games," Proc. DAC, 2008.

[4] T. Sherwood, *et al.*, "Discovering and Exploiting Program Phases," IEEE Micro, Nov/Dec, 2003.

[5] T. Sherwood, *et al.*, "Phase Tracking and Prediction," Proc. ISCA, 2003.

[6] Q. Wu, *et al.*, "A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance," Proc. MICRO, 2005.

[7] C. Isci, *et al.*, "Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management," Proc. MICRO, 2006.

[8] S. Lee and T. Sakurai, "Run-time Voltage Hopping for Low-Power Real-time Systems," Proc. DAC, 2000.

[9] A. Azevedo, *et al.*, "Profile-based Dynamic Voltage Scheduling using Program Checkpoints," Proc. DATE, 2002.

[10] D. Shin and J. Kim, "Optimizing Intra-task Voltage Scheduling using Data Flow Analysis," Proc. ASPDAC, 2005.

[11] J. Seo, *et al.*, "Profile-based Optimal Intra-task Voltage Scheduling for Hard Real-time Applications," Proc. DAC, 2004.

[12] S. Hong, *et al.*, "Runtime Distribution-aware Dynamic Voltage Scaling," Proc. ICCAD, 2006.

[13] S. Hong, *et al.*, "Dynamic Voltage Scaling of Supply and Body Bias Exploiting Software Runtime Distribution," Proc. DATE, 2008.

[14] J. Kim, *et al.*, "An Analytical Dynamic Volage Scaling of Supply Voltage and Body Bias Based on Parallelism-aware Workload and Runtime Distribution," to appear in IEEE Transactions on CAD.

APPENDIX

If we substitute w_i with Eqn. (4) in Eqn. (3) and if we assume that the PDF of n_i is represented by M bins, i.e., M pairs of execution cycle and probability, $< x_i(k), p_i(k) >$'s, we obtain the following equation.

$$\overline{x_i} + (w_{i+1}^{eff})^{b+1} \cdot [\sum_{k=1}^{M} \frac{-x(k)p_i(k)}{(x_i^{eff} + w_{i+1}^{eff} - x_i(k))^{b+1}}] = 0$$

As shown in the above equation, x_i^{eff} depends on $\overline{x_i}$, w_{i+1}^{eff}, and the PDF of n_i, i.e., $< x_i(k), p_i(k) >$'s.