Application-Support Particle Filter for Dynamic Voltage Scaling of Multimedia Applications

Jae-Beom Lee, Myoung-Jin Kim, Sungroh Yoon, *Senior Member*, *IEEE*, and Eui-Young Chung, *Member*, *IEEE*

Abstract—*Dynamic Voltage and Frequency Scaling* (DVFS) is an effective low-power technique for real-time workloads. Its effectiveness critically depends on the accurate prediction of the task execution time. Many DVFS approaches have been proposed, but they are insufficient for highly nonstationary workloads. Several recent DVFS techniques adopted adaptive filters to improve accuracy. However, their improvement was rather limited, since they mainly focused on applying a filter framework to the target application without tuning it. We address this issue by proposing *Particle Filter* (PF)-based video decoders (MPEG2 and H.264) which exploit application-specific characteristics. More specifically, our PF-based video decoders utilize the size of each frame for the prediction of its decoding time. Compared to previous work, the PF is more suitable for our purpose, since it achieves higher prediction accuracy, even for highly nonstationary workloads such as H.264 clips. Our results show that the energy saved by the proposed approach is comparable to that of the ideal policy called *oracle-DVFS*, while the existing methods we tested were far inferior to oracle-DVFS in terms of H.264 video decoding. Additionally, when our method was used, only 0.40 and 6.88 percent of the frames missed their deadlines with negligible computational overhead for MPEG and H.264, respectively.

Index Terms—Dynamic voltage and frequency scaling, feedback control, low energy, sequential Monte Carlo, particle filter, nonstationarity.

1 Introduction

Low-power design has become a key challenge due to the widespread use of battery-powered portable multimedia devices, such as cellular phones, PDAs, MP3 players, and so on. Dynamic Power Management (DPM) and Dynamic Voltage and Frequency Scaling (DVFS) are popular system-level low-power techniques for these devices.

DPM aims at reducing the power consumption of a system by shutting it down when it is idle [1]. On the other hand, DVFS aims at reducing the energy consumed by a task running on a system by scheduling voltage/frequency (v/f) pairs such that the task is completed within the given deadline constraints. DVFS is known as one of the most effective low-power techniques, especially for real-time embedded systems. In DVFS, it is very important to predict the task execution time accurately, in order to minimize the energy consumption (EC) while satisfying the given deadline constraints. In many real-time embedded systems, a task has a periodic nature, but its execution time at each period significantly varies due to the workload variations.

In hard real-time applications, workload variations are simply resolved by introducing the concept of the Remaining Worst case Execution Path (RWEP) [3], which often requires extensive profiling. RWEP-based DVFS techniques are conservative in the sense that deadline satisfaction is considered more important than energy savings. On the other hand, soft real-time applications are allowed to violate the given deadline constraints within a tolerable range and, hence, they can adopt more aggressive techniques to reduce the energy consumption. Many of the previous DVFS techniques focused on video decoding, since it is one of the most popular soft real-time applications in power-conscious portable devices. For this reason, we limit our discussion in this paper to multimedia applications, especially those used for video decoding.

Several variants of RWEP have been proposed for soft real-time applications by considering the average case or oracle case rather than the worst case [4], [5]. These methods profile the execution time and frequency of the code sections in a target application and predict the execution time based on the control flow graph (CFG) annotated with the execution probability of the code sections. Another group of heuristic methods empirically derive a workload predictor from application-specific features [6], [7]. The well-known techniques in this category utilize the frame size to estimate the task execution time (i.e., frame decoding time in this case) by profiling and linear regression. Generally speaking, profiling-based methods statically set the parameters of their workload predictors during design time. Hence, their effectiveness can be severely degraded for unexpected workloads.

To overcome the limitations of profiling-based methods, several online methods have been proposed. They can be classified into three main categories. The techniques in the first category nondeterministically make a decision on v/f pair selection based on the workload

- *J.-B. Lee, M.-J. Kim, and E.-Y. Chung are with the School of Electrical and Electronic Engineering, Yonsei University, 134 Sinchon-dong, Seodaemun-gu, Seoul 120-749, Korea. E-mail: {jblee, kmjjang86, eychung}@dtl.yonsei.ac.kr.*
- *S. Yoon is with the Department of Electrical and Computer Engineering, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul 151-744, Korea. E-mail: sryoon@gmail.com.*

Manuscript received 20 Oct. 2010; revised 28 June 2011; accepted 20 July 2011; published online 1 Aug. 2011.
Recommended for acceptance by E. Macii.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2010-10-0579.
Digital Object Identifier no. 10.1109/TC.2011.148.

statistics collected during runtime [8], [9]. Their effectiveness is limited to stationary workloads. The approaches in the second category are similar to profiling-based empirical methods. The only difference is that they provide an online parameter tuning scheme, but they still have the same shortcomings as the empirical methods have [7]. The methods in the final category utilize statistical adaptive filters [18], [19], [20]. They estimate future workloads based on the workload history in a statistical framework. More advanced methods in this category adaptively correct prediction errors using a feedback mechanism. These methods show nonmarginal improvements over the aforementioned methods, but their effectiveness is still not sufficient for highly nonstationary workloads.

In this paper, we propose a filter-based online method which exploits some useful application characteristics to handle highly nonstationary workloads. Our method adopts an adaptive nonlinear filter called the *Particle Filter* (PF) [10], [11] and customizes it for video decoding based on the results obtained from previous empirical methods. More specifically, we utilize the frame size as an important application characteristic for predicting its decoding time. In our framework, we first estimate a function which correlates the frame size and its decoding time for each frame. Then, the function is fed to the PF to estimate the frame decoding time. In other words, the linear function roughly estimates the decoding time first and then the PF refines the estimates using its error-covariance feature. The proposed method was implemented in both MPEG2 and H.264 players and was validated on an ARM-based real testbed.

The rest of this paper is organized as follows: related work is presented in Section 2. The motivation of our work is presented in Section 3. In Section 4, we provide a brief summary of the particle filter. We propose a PF-based DVFS technique for real multimedia applications such as MPEG and H.264/AVC decoders in Section 5. Finally, we show the effectiveness of our approach by comparing it with other estimators in Section 6, followed by our conclusion in Section 7.

2 Related Work

DVFS techniques can be classified into three categories as briefly addressed in Section 1. We will discuss further details of these categories in this section.

2.1 DVFS Techniques with Execution Time Profiling

One of the earliest DVFS techniques presented in [12] proposed an optimal v/f pair scheduling technique under the assumption that both the task arrival time and execution time are constant. Later, many profiling-based techniques were proposed to consider workload variations using the concepts of the Remaining Worst case Execution Path, Remaining Average-case Execution Path (RAEP), and Remaining Oracle-case Execution Path (ROEP). The approaches presented in [3], [4], and [5] performed path-based profiling for the control flow graph of a target task and extracted the RWEP, RAEP, and ROEP, respectively. These techniques scheduled v/f pairs at several control points inside a task based on some profiled statistics. Other task-level DVFS techniques also used the RWEP, RAEP, and ROEP concepts with profiled statistics [13], [14]. The limitations of these methods are twofold. First, their accuracy cannot be guaranteed when they encounter workloads different from the profiling statistics. Second, they assume that the workloads are stationary, which means that the parameters characterizing them are constant or time invariant, even when they are really time varying.

The more advanced techniques proposed in [8], [9] performed intratask-level v/f pair scheduling using a probabilistic distribution function (PDF) or cumulative distribution function (CDF), which can be collected during runtime. Even though these methods provide online workload statistics, they are still appropriate mostly for stationary workloads.

2.2 DVFS Techniques with Application Characteristics

Another group of methods were designed especially for multimedia applications. They estimated the workloads by exploiting the correlation between the frame size and the execution time [6], [15], [16], [17]. In [6], the authors estimated the MPEG frame decoding time using a linear function of the frame data sizes. The coefficients of the linear function are obtained through offline profiling; hence, its accuracy critically depends on the training data used. In [17], the authors divided the frame decoding time into the frame-independent (FI) part and frame-dependent (FD) part, where the FI part was simply estimated using a linear function of the number of macro blocks, while the FD part was estimated from the moving average (MA) of the FD parts of the previous frames. The authors in [15] focused on the hierarchical layered structure in MPEG video streams. Each frame is further divided into slices, each of which contains several macro blocks. As in the case of frames, there are different types of macro blocks. Different types of blocks require different processing times during decoding. A linear estimation function is designed for each block type based on profiling and the overall frame decoding time is estimated by summing the estimations for all types of blocks. Even though this method showed impressive results in certain cases, it still needs profiling and requires the parameters to be tuned during design time. Also, the estimation accuracy (EA) shows a large variation depending on the nature of video clips.

2.3 Filter-Based DVFS Techniques

The methods in the final category are based on statistical adaptive filters. They attempt to follow the variation of the workloads during runtime. The work in [18] adopted simple filters such as the *moving average* and *weighted moving average* (WMA) filters. The MA filter predicts the workload execution time in the next slot as the average time of the workload in the previous N slots, while the WMA filter uses the weighted mean of the previous workloads to estimate the next slot. Their accuracy and tracking speed show a trade-off relationship with the number of previous slots considered. Even though these methods are simple enough to be applicable to DVFS, their estimation accuracy is limited, due to the absence of an error-correcting feedback mechanism.

A *proportional-integral-derivative* (PID) controller was also used for this purpose [19]. The PID controller is a generic recursive filter widely used in many applications. It adjusts the system parameters based on the feedback from the recent error between a measured process variable and a desired set point. However, it is known that its control policy is highly sensitive to the tuning of the coefficients used. The methodology in [20] employed a workload estimation technique using the Kalman filter [2]. However, the Kalman filter is usually used for linear-Gaussian cases and, thus, its efficiency may be severely degraded when highly nonstationary workloads need to be processed. Even though filter-based techniques show nonmarginal improvement over the other techniques, their efficiency is still unsatisfactory for highly nonstationary workloads. This is because they focus only on applying the filters to DVFS without tuning their parameters using useful application characteristics.

2.4 Memory-Aware DVFS Techniques

Some of previous works further advanced DVFS techniques by decomposing the decoding process into two different subprocesses: CPU-bound work and memory-bound work [21], [22], [23]. These approaches can select v/f pairs more aggressively by recognizing that CPU is idle, while the memory-bound work is performed. The effectiveness of these approaches also depends on the accuracy of workload estimations for CPU-bound work and memory-bound work, respectively. Hence, the accurate workload estimation is still essential for these approaches.

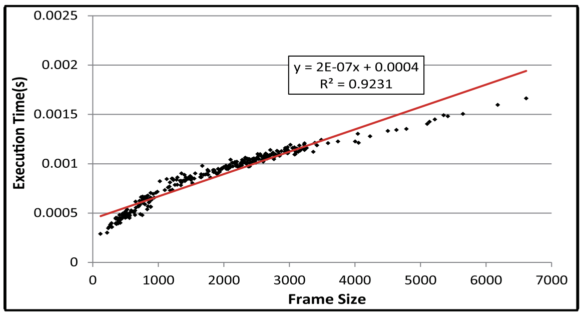
2.5 Contributions

The proposed method tackles the issues of existing methods by customizing an adaptive filter to exploit some useful application-specific characteristics. It has some resemblance to the PID controller-based and Kalman filter-based techniques, since we also adopt an adaptive filter called the *Particle Filter* [11]. However, the PF is more powerful than the PID controller and Kalman filter in the sense that it can handle even nonlinear/non-Gaussian time-varying workloads [10]. Furthermore, our PF is customized for video decoding by exploiting some useful application characteristics. The major goal of this work is to propose an accurate workload estimation method; hence, we do not distinguish the CPU-bound work and memory-bound work in this paper. However, it can be easily extended for this purpose, since the proposed estimation method is generically presented in a vectored form.

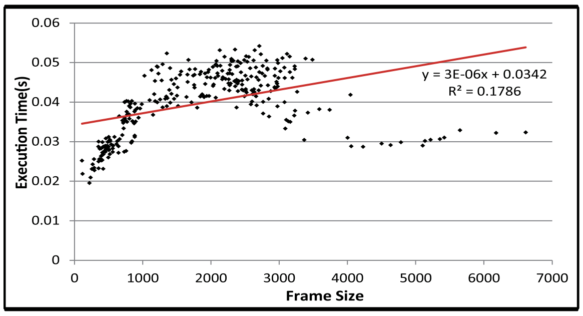
3 Motivation

The processing time of multimedia workloads, especially those that have a large variation in their decoding time, is the focus of this paper. There exist many standards for multimedia applications, e.g., MPEG-2/4, H.264/AVC, and MP3. Among these, we consider two popular video formats—MPEG-2 and H.264, since their workloads are known to typically have large variations in their execution time. The time-varying property of their workloads is the largest obstacle to the application of DVFS to low-power multimedia applications, since the quality of DVFS critically depends on the prediction accuracy of the frame decoding time.

The workloads we use consist of a series of picture frames. There are three types of picture frames defined in the MPEG and H.264 standards [27]. Each of the frames is



(a)



(b)

Fig. 1. Relationships between frame size and execution time. (a) MPEG decoding. (b) H.264 decoding.

compressed in a different manner for lossy compression. Even though lossy compression improves portability, it increases the time-varying property of the video sequences. Hence, the accurate prediction of the frame decoding time has become a more challenging task in DVFS.

We were inspired to integrate two well-known approaches to the DVFS of multimedia applications in order to capitalize on their strong points. The first one is from a group of empirical methods which build regression models based on profiling for the purpose of estimating the frame decoding time with respect to the frame size. Fig. 1 shows the relationships between the frame size and the execution time obtained from one of these empirical methods. The empirical method used in Fig. 1 correlates the decoding time and the frame size in a linear fashion, where Figs. 1a and 1b correspond to MPEG and H.264, respectively.

Fig. 1 shows that the empirical method is a feasible solution for MPEG. However, it is not a feasible method in the case of H.264, since the correlation between the frame size and the frame decoding time is weak relative to the case of MPEG. The weak correlation in H.264 is mainly due to the more aggressive encoding used in H.264 compared to MPEG. Obviously, we need a higher order function to improve the estimation accuracy for H.264. Moreover, the accuracy will be further improved if we provide a mechanism which enables us to adjust the decoding time variation in conjunction with the higher order function.

There is also a critical issue common to MPEG and H.264. The model is fitted based on profiling, meaning that the parameters should be tuned during the design time. Hence, this may cause severe estimation errors when it encounters a video clip which has a completely different nature from the training clips. In other words, the model should be transformed into an online version to track the nature of the video clip being decoded.

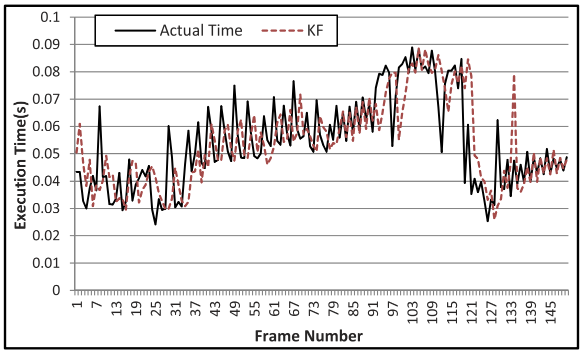


Fig. 2. Phase delay between actual decoding time and estimated decoding time of the Kalman filter.

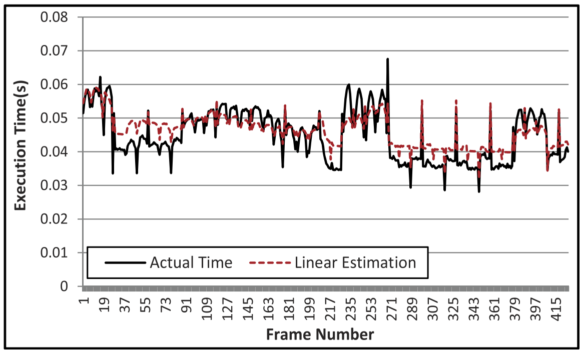


Fig. 3. Estimation of the linear regression method.

The methods in the third group are statistical approaches which mostly adopt adaptive filters (e.g., the Kalman filter) to capture the time-varying nature of the video stream. The strong point of these methods is that they are more robust to workload variations than empirical methods, since they are capable of correcting the estimation error thanks to their error-feedback control mechanism.

The pros and cons of the two groups are clearly depicted in Figs. 2 and 3. Fig. 2 compares the actual decoding time and the time estimated by the Kalman filter for a video clip in H.264. Similarly, Fig. 3 compares the actual decoding time and the time estimated by a linear function for the same video clip. One interesting observation in Fig. 2 is that the Kalman filter tracks the actual decoding time well in terms of its magnitude, but with a certain amount of delay. The delay becomes critical to ensuring the estimation accuracy when the actual decoding time of a certain frame abruptly changes compared to those of its neighboring frames. In this case, the magnitude of the estimation significantly deviates from the actual decoding time, since the change in the actual decoding time is faster than the adaptation speed of the filter. Intuitively, any adaptive method has a delay effect. On the other hand, the estimation by the linear function shows somewhat different characteristics in Fig. 3. Its estimation is always in phase with the actual decoding time. However, there is a certain amount of discrepancy between the magnitudes of the estimated and actual decoding times. This is because the linear model (as well as other nonstatistical models) cannot consider the variance of its input and output.

These examples motivated us to integrate the adaptive filter-based method with the empirical method such that the delay effect of an adaptive filter is compensated by the empirical model, while retaining its accuracy in terms of the magnitude. More precisely, to estimate current execution time, our PF utilizes the current frame size information to estimate the frame decoding time within the PF framework, as in the case of the linear regression method, and then refines the estimate using its error-covariance feature, by acting as another adaptive filter. In other words, it first estimates the decoding time using the linear regression method and, then, removes the error by means of using the PF. As a result, our PF guarantees a higher estimation accuracy compared to other filter-based methods. To achieve this, we identify three challenging issues as follows:

- We need an online regression model to estimate the frame decoding time with respect to the frame size. This model should be suitable for workloads which have nonlinear characteristics such as H.264.
- We also need an adaptive filter with an error-control feedback mechanism. This filter will track the time-varying property of the workloads, while the aforementioned estimator captures the other time-independent properties of the workloads.
- We need to integrate the two estimators mentioned above with negligible computational overhead.

More details are given in Section 5.

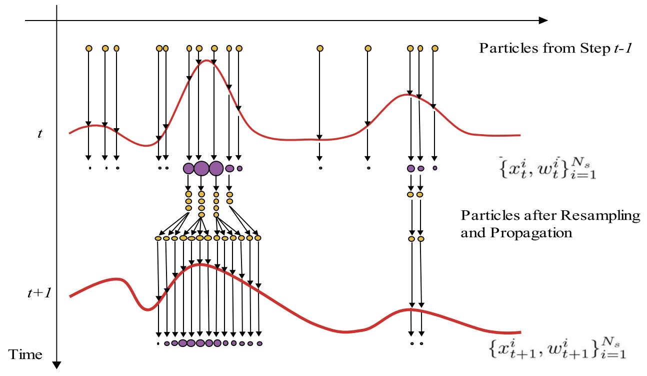
4 Generic Particle Filter

4.1 Overview

In most real-time applications, prior knowledge about the phenomenon being modeled is available. This knowledge allows us to formulate Bayesian models, where all inferences on unknown quantities are based on the posterior distribution obtained from the Bayes' theorem [11]. However, real-time applications can be very complex, involving elements of non-Gaussianity, high dimensionality, and nonstationarity. To handle these complications, many different filters have been proposed. The *sequential Monte Carlo* (SMC) [28] method is a set of simulation-based methods which provide a convenient and attractive approach to computing the posterior distribution. The *particle filter* [11] is closely related to the SMC method and is a technique for implementing a recursive Bayesian filter. The main idea is that the PF represents the posterior density by a set of random masses (called *particles*) with associated weights and performs estimation based on these particles and weights [10].

The particle filter algorithm operates recursively in two stages: *prediction* and *update*. The prediction stage is to modify each particle according to the existing system model. For instance, to simulate the noise effect on the variable, random noise is added to each particle. The update phase is to reevaluate the weight of each particle based on the latest measurement made. This process is also known as the *sequential importance sampling* (SIS).

Usually, after a few recursive steps, all but a few particles will have negligible weights. This issue is called the *degeneracy phenomenon* and should be handled appropriately, since it has a harmful effect on the accuracy of the PF. To resolve this issue, the PF needs to resample the particles. In the resampling step, the PF discards the samples with



Particles from Step t-1
t
$\{x_t^i, w_t^i\}_{i=1}^{N_s}$
Particles after Resampling and Propagation
t+1
Time
$\{x_{t+1}^i, w_{t+1}^i\}_{i=1}^{N_s}$

Fig. 4. A pictorial description of the particle filter.

low importance weights and multiplies those with high importance weights. Of note is that the effect of the degeneracy problem was rather marginal in this study, due to the large variations of the workloads.

A graphical representation of a PF is shown in Fig. 4 [24]. This figure shows the basic idea of the particle filter that represents the posterior density by a set of random particles with associated weights. At the top, the PF starts with uniformly weighted particles (the top yellow dots) which approximate the prediction density (the top red line). Then, the PF computes the importance weight of each particle (the top purple dots, $\{x_t^i, w_t^i\}_{i=1}^{N_s}$). In this situation, the resampling step is executed, because all but a few particles will have negligible next estimation values. The last step is the prediction based on weighted particles. This process is executed recursively.

4.2 Generic Particle Filter Details

The objective of particle filtering is to track a time-evolving variable of interest. To define this more formally [10], consider the evolution of the state sequence $\{\mathbf{x}_t, t = 0, 1, \ldots\}$. Here, $\mathbf{x}_t$ is a vector of states at time t given by

$$\mathbf{x}_t = f_t(\mathbf{x}_{t-1}, \mathbf{v}_{t-1}), \tag{1}$$

where f_t is a (possibly nonlinear) function of $\mathbf{x}_{t-1}$ and $\mathbf{v}_{t-1}$, an i.i.d. process noise at time $t-1$. The objective of tracking is then to estimate $\mathbf{x}_t$ recursively from measurements

$$\mathbf{z}_t = h_t(\mathbf{x}_t, \mathbf{n}_t), \tag{2}$$

where h_t is a (possibly nonlinear) function of $\mathbf{x}_t$ and $\mathbf{n}_t$, an i.i.d. measurement noise. We are particularly interested in filtered estimates of $\mathbf{x}_t$ based on the measurement sequence $\mathbf{z}_{1:t} = \{\mathbf{z}_k, k = 1, 2, \ldots, t\}$ up to time t. These two equations are known as the *state* and *measurement* equations, respectively.

The tracking problem from a Bayesian perspective requires constructing the pdf $p(\mathbf{x}_t|\mathbf{z}_{1:t})$. Let $\{\mathbf{x}_t^i, w_t^i\}_{i=1}^{N_s}$ (N_s is the number of particles) denote a *random measure* that characterizes the posterior pdf $p(\mathbf{x}_t|\mathbf{z}_{1:t}$, where $\{\mathbf{x}_t^i, i = 0, \ldots, N_s\}$ is a set of support points with associated weights $\{w_t^i, i = 1, \ldots, N_s\}$. The weights are normalized such that $\sum_i w_t^i = 1$. Using the principle of importance sampling [11], we can calculate the weights as follows:

$$w_t^i \propto w_{t-1}^i \frac{p(\mathbf{z}_t|\mathbf{x}_t^i) p(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i)}{q(\mathbf{x}_t^i|\mathbf{x}_{t-1}^i, \mathbf{z}_t)}, \tag{3}$$

where $q(\cdot)$ is a proposal distribution called an *importance density* [10]. Filtering via a PF thus consists of the recursive propagation of the importance weights w_t^i and particles x_t^i as each measurement is received continuously.

The choice of the importance density plays a crucial role in the overall design. We can derive the optimal importance density and weights from the SIS algorithm as follows [10]:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}^i, \mathbf{z}_t) = \frac{p(\mathbf{z}_t|\mathbf{x}_t, \mathbf{x}_{t-1}^i) p(\mathbf{x}_t|\mathbf{x}_{t-1}^i)}{p(\mathbf{z}_t|\mathbf{x}_{t-1}^i)}. \tag{4}$$

Plugging (4) into (3) yields

$$w_t^i \propto w_{t-1}^i p(\mathbf{z}_t|\mathbf{x}_{t-1}^i), \tag{5}$$

which states that the importance weights at time t can be computed before particles are propagated to time t.

To solve the degeneracy problem, a suitable measure of an algorithm is the effective sample size $\hat{N}_{eff}$ introduced in [29], which is estimated as follows:

$$\hat{N}_{eff} = \frac{1}{\sum_{i=1}^{N_s} (w_k^i)^2}. \tag{6}$$

According to this equation, a small $\hat{N}_{eff}$ indicates severe degeneracy and vice versa. Also, the threshold, N_{th}, is set by the user. To handle the degeneracy problem, if $\hat{N}_{eff}$ is lower than N_{th}, we invoke the resampling algorithm to avoid it. Resampling involves the mapping of the random measure $\{\mathbf{x}_t^i, w_t^i\}$ into a random measure $\{\mathbf{x}_t^i, 1/N_s\}$ with uniform weights.

The marginal distribution $p(\mathbf{x}_t|\mathbf{z}_t)$ satisfies the following recurrence formula:

- *Update*:

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}) = \frac{p(\mathbf{z}_t|\mathbf{x}_t) p(\mathbf{x}_t|\mathbf{z}_{1:t-1})}{\int p(\mathbf{z}_t|\mathbf{x}_t) p(\mathbf{x}_t|\mathbf{z}_{1:t-1}) d\mathbf{x}_t}. \tag{7}$$

- *Prediction*:

$$p(\mathbf{x}_t|\mathbf{z}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1}) d\mathbf{x}_{t-1}. \tag{8}$$

This general and simple algorithm forms the basis of most particle filters.

Consider the case where 1) the state function is represented by a nonlinear polynomial equation, 2) the measurement equation is linear (in this work, H_k is unity because an estimation value is mapped to an observation value), and 3) all the random particles in the model are additive Gaussian because there is no information on the distribution of decoding time. Such a system is given by [10], [11]:

$$\mathbf{x}_t = f_{t-1}(\mathbf{x}_{t-1}) + \mathbf{v}_{t-1}, \tag{9}$$

$$\mathbf{z}_t = \mathbf{H}_t \mathbf{x}_t + \mathbf{n}_t, \tag{10}$$

where $\mathbf{v}_{t-1}$ and $\mathbf{n}_t$ are mutually independent zero-mean white Gaussian with covariances $\mathbf{Q}_{t-1}$ and $\mathbf{R}_t$, respectively.

```
Algorithm I. Generic Particle-filter
input   number of particles $N_s$
        previous sets of particles $\mathbf{x}_{t-1}$ and weights $\mathbf{w}_{t-1}$
output  current sets of particles $\mathbf{x}_t$ and weights $\mathbf{w}_t$
 1: begin
 2:     for (i = 1 ; i < $N_s$ ; i++)
 3:         draw the particles $x_t^i \sim p(\mathbf{x}_k|x_{t-1}^i)$
 4:         evaluate the importance weights, Eq. (5);
 5:     end for
 6:     calculate total weight $k = \sum_{i=1}^{N_s} w_t^i$;
 7:     for (i = 1 ; i < $N_s$; i++)
 8:         normalize $w_t^i = w_t^i / k$;
 9:     end for
10:     estimated value $\hat{x}_t = \sum_{i=1}^{N_s} x_t^i \cdot w_t^i$
11:     if $\hat{N}_{eff}$ is smaller than $N_{th}$ then
12:         resample;
13:     end if
14: end
```

Fig. 5. Generic particle filter.

Then, the optimal importance density and $p(\mathbf{z}_t|\mathbf{x}_{t-1})$ are also Gaussian:

- *Update*:

$$p(\mathbf{z}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{z}_t; \mathbf{b}_t, \mathbf{S}_t), \tag{11}$$

$$\mathbf{S}_t = \mathbf{H}_t \mathbf{Q} \mathbf{H}_t^T + \mathbf{R}, \tag{12}$$

$$\mathbf{b}_t = \mathbf{H}_t f_{t-1}(\mathbf{x}_{t-1}). \tag{13}$$

- *Prediction*:

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{z}_t) = \mathcal{N}(\mathbf{x}_t; \mathbf{a}_t, \mathbf{\Sigma}_t), \tag{14}$$

$$\mathbf{a}_t = f_{t-1}(\mathbf{x}_{t-1}) + \mathbf{\Sigma}_t \mathbf{H}_t^T \mathbf{R}^{-1}(\mathbf{z}_t - \mathbf{b}_t), \tag{15}$$

$$\mathbf{\Sigma}_t = \mathbf{Q} - \mathbf{Q}\mathbf{H}_t^T \mathbf{S}_t^{-1} \mathbf{H}_t \mathbf{Q}, \tag{16}$$

where $\mathcal{N}(\cdot)$ is a Gaussian distribution represented by its mean and covariance and $\mathbf{a}_t$ and $\mathbf{\Sigma}_t$ are the mean and covariance of (14), respectively. Also, $\mathbf{b}_t$ and $\mathbf{S}_t$ are mean and covariance of (11), respectively. On the other hand, $\mathbf{H}_k$ and $f_k(\cdot)$ are an observation matrix and a (possibly nonlinear) state function, respectively. $\mathbf{Q}$ and $\mathbf{R}$ are the update and measurement error covariances, respectively.

Fig. 5 outlines the generic particle filter algorithm which is widely used [10]. The generic particle filter might be applied to a DVFS technique for multimedia applications as it is. Two key steps of a generic particle filter are the sequential importance sampling and resampling. There are many more types of particle filters and many other variants have been proposed. Further optimizations are possible by reflecting the domain knowledge. The proposed PF-based DVFS algorithm for multimedia decoding will be presented in Section 5. In this section, we provide some preliminary materials to facilitate the explanation in Section 5. For a more thorough treatment of this topic, the reader is directed to [10]. Finally, the estimated value, $\hat{x}_t$, is calculated by $\sum_{i=1}^{N_s} x_t^i \cdot w_t^i$.

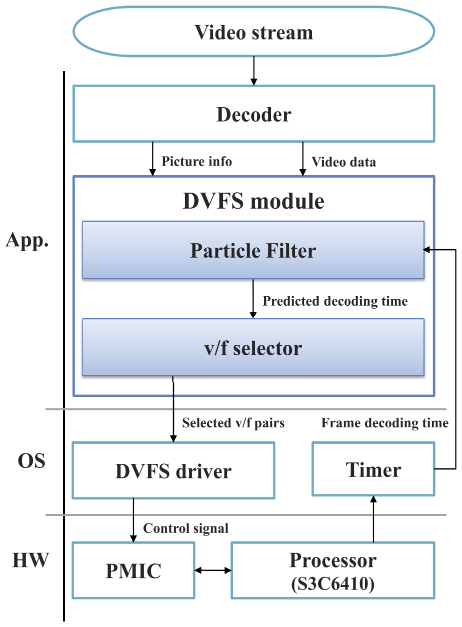


Fig. 6. Overall structure of proposed method.

To summarize, the PF can handle the estimation of even nonlinear/non-Gaussian workloads. Its estimation accuracy critically depends on how $f_k(\cdot)$, $h_k(\cdot)$, $\mathbf{Q}$, and $\mathbf{R}$ are defined. We define $f_k(\cdot)$ using the relationship between the frame decoding time and frame size, while setting $\mathbf{Q}$ and $\mathbf{R}$ based on the past history of observations and measurements, respectively, as will be discussed in Section 5.

5 Proposed Method

5.1 Overall Structure

The overall structure of our method is shown in Fig. 6. Once a video stream is transmitted to a client system, it decodes the frames in their transmission order. In the case of software video decoders, they typically consist of three layers. In the application layer, there is a software video decoder for decoding the video streams. In the OS layer, there is a DVFS driver which sends an appropriate v/f pair to the power management IC (PMIC) in the HW layer for the purpose of controlling the voltage and frequency levels of the target processor. Also, a timer driver can be utilized for the estimation of the decoding time. Note that the timer measures the execution cycles of each frame decoding rather than the decoding time. Once it is fed to the particle filter, it is multiplied by the inverse of the current clock frequency to check the deadline miss. Also, it is multiplied by the inverse of the maximum clock frequency for being used in the next frame decoding estimation. In our case, there is an additional SW module called *DVFS module* for supporting DVFS in the application layer. *DVFS module* internally has two submodules—*Particle filter* and *v/f selector* (shaded modules in Fig. 6). The PF estimates the frame decoding time with its *Updater* and *Predictor*; *v/f selector* in the *DVS module* chooses an appropriate v/f pair based on the estimation from the PF.

5.2 Tuning the Parameters of PF

To apply the PF to DVFS-support video decoders, we first tune its essential parameters, as discussed in Section 4.2.

The focus of this section is how to customize the PF for video decoders based on the relationship between the frame size and decoding time.

There are three types of frames in multimedia applications. We dedicate one particle filter to each frame type in order to easily track the workload variations. The parameters of these PFs are identical in their formulation, but their values may differ, since they would experience different workloads. More specifically, we want to customize the parameters (state function $f_t(\cdot)$, observation matrix $\mathbf{H}_t$, estimation error covariance $\mathbf{Q}$, and measurement error covariance $\mathbf{R}$) of (11) through (16) by correlating the frame decoding time with the frame size. The coefficients of these parameters are dynamically updated during the decoding process. In other words, they are time-varying parameters in nature. The observation matrix, $\mathbf{H}_t$, is exceptionally set to unity in our method, since we can always know the exact observation value by measuring the decoding time using a timer.

To summarize, the state and measurement equations used are (9) and (10), respectively. We present more details of the state function f_t in Section 5.2.1. Section 5.2.2 explains how the covariances $\mathbf{Q}$ and $\mathbf{R}$ of the variables $\mathbf{v}_{t-1}$ and $\mathbf{n}_t$ in (9) and (10) are computed. The observation matrix $\mathbf{H}_t$ is set to unity as explained above.

5.2.1 State Function $f_t(\cdot)$

The state function $f_t(\cdot)$ provides a rough estimate of the frame decoding time when a frame is decoded. This rough estimate is finely tuned by the other parameters ($\mathbf{R}$ and $\mathbf{Q}$) based on the error-correcting feedback mechanism.

We define the state function $f_t(\cdot)$ in a polynomial form, since a polynomial can easily be adjusted by controlling its order and coefficients. The main challenge in defining $f_t(\cdot)$ is twofold. First, we need to determine an appropriate order of a polynomial to achieve reasonable estimation accuracy. Second, we need to minimize the computational overhead in updating its coefficients, since they are updated during runtime (online) in our method. The accuracy and computational overhead are usually inversely related to respect to the order of the polynomial. In other words, a higher order polynomial improves the accuracy at the expense of a significant increase in the computational overhead.

To resolve this issue, we start from a first-order polynomial function, as shown in (17) and then explore higher order functions to identify the best polynomial function for our purpose.

$$f_t(s) = c_1 s + c_0. \tag{17}$$

In (17), s is the frame size and $f_t(s)$ is the estimated decoding time for the given frame size at time t. We need to determine two coefficients c_1 and c_0 such that the difference between $f_t(\cdot)$ and the actual decoding time (x_t) is minimized, in order to maximize the estimation accuracy. If a PF experiences n frames up to time t, then the estimation error can be written as

$$error = \sum_{i=1}^{n} (x_t(s_i) - f_t(s_i))^2, \tag{18}$$

$$= \sum_{i=1}^{n} (x_t(s_i) - (c_1 s_i + c_0))^2. \tag{19}$$

To minimize the error shown in (19), we take the derivatives of (19) with respect to c_1 and c_0, respectively, and then set each of them to zero:

$$\frac{\partial error}{\partial c_1} = -2\sum_{i=1}^{n} (x_i - c_1 s_i - c_0) s_i = 0, \tag{20}$$

$$\frac{\partial error}{\partial c_0} = -2\sum_{i=1}^{n} (x_i - c_1 s_i - c_0) = 0. \tag{21}$$

Representing these two equations in a matrix form yields

$$\begin{bmatrix} \sum {s_i}^2 & \sum s_i \\ \sum s_i & n \end{bmatrix} \begin{bmatrix} c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} \sum s_i x_i \\ \sum x_i \end{bmatrix}. \tag{22}$$

For notational simplicity, let us denote each element of the matrix as a single variable, namely, $i_n = \sum_{i=1}^{n} s_i^2$, $j_n = \sum_{i=1}^{n} s_i$, $k_n = \sum_{i=1}^{n} s_i x_i$, and $l_n = \sum_{i=1}^{n} x_i$.

Then, we can rewrite these elements in a recursive manner based on the results in the previous time step as follows:

$$i_n = i_{n-1} + s_n^2, \tag{23}$$

$$j_n = j_{n-1} + s_n, \tag{24}$$

$$k_n = k_{n-1} + s_n x_n, \tag{25}$$

$$l_n = l_{n-1} + x_n. \tag{26}$$

Equations (23) through (26) are more appropriate for online estimation, since they can greatly reduce the computation overhead by reusing the previous results.

After plugging the above variables for the variables into (22), we can rearrange (22) with respect to c_1 and c_0. Then, c_1 and c_0 for n can be written as

$$c_1 = \frac{n k_n - j_n l_n}{n i_n - (j_n)^2}, \tag{27}$$

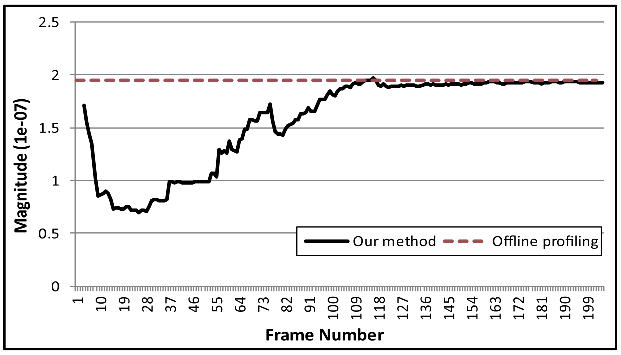
$$c_0 = \frac{i_n l_n - j_n k_n}{n i_n - (j_n)^2}. \tag{28}$$

The state function is updated using (27) and (28), whenever a frame is decoded.

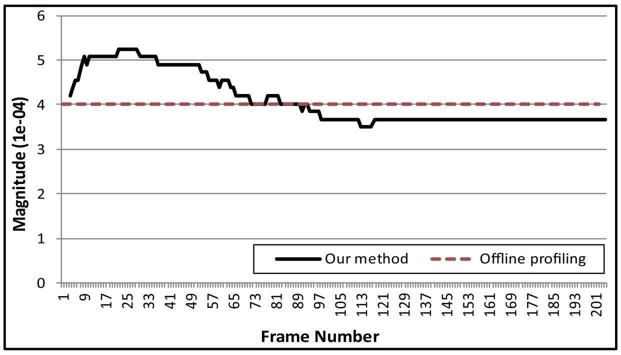
We can generalize the state function to consider a higher order polynomial such that $f_t(s) = \sum_{i=0}^{n} c_i s^i$. Then, we can obtain higher order polynomial coefficients in a recursive manner as we did for the first-order polynomial.

To assess the effectiveness of the aforementioned online version of the state function, we first investigated the estimation accuracy of the first-order state function by measuring the adaptation speed of its coefficients. As shown in Fig. 7, both coefficients (c_1 and c_0) converge to the values obtained from the offline analysis after experiencing about 100 frames. Using higher order state functions also produced similar results.

Next, we analyzed the estimation accuracy and computation overhead of the state function with respect to its



(a)



(b)

Fig. 7. Adaptation speed of coefficients of state function in online environment. (a) c_1. (b) c_0.

order, as shown in Fig. 8. Obviously, the estimation accuracy is improved as the order of the state function increases, while incurring more computational overhead. We take the first-order polynomial as a state function since its mean square error (MSE) is small enough to achieve the reasonable accuracy while paying small computational overhead. However, higher order polynomials can be used for future video codecs such as H.264 since they show more nonlinear behavior in video decoding. In contrast, the linear filters including the Kalman filter cannot handle the highly nonlinear behavior expected in future video codecs.

Choosing the first-order polynomial may incur severe accuracy degradation of the state function, especially for H.264, since it shows weak correlation between the frame size and the decoding time compared to MPEG.

However, notice that the degradation of the accuracy incurred by choosing a lower order polynomial for the state function is compensated by the other parameters ($\mathbf{Q}$ and $\mathbf{R}$) which perform the fine-tuning of the estimation based on the error-correcting feedback mechanism provided by PF. More detailed experimental results will be discussed in Section 6.

5.2.2 Error Covariance, $\mathbf{Q}$ and $\mathbf{R}$

The estimation error covariance ($\mathbf{Q}$) and measurement error covariance ($\mathbf{R}$) finely tune the rough estimation from the state function $f_t(\cdot)$. More precisely, $\mathbf{Q}$ adjusts the current estimate by comparing it with the previous estimates. Similarly, $\mathbf{R}$ adjusts the current estimate by comparing it with the previous measurements.

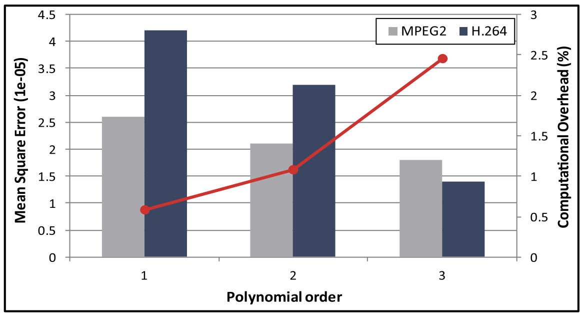


Fig. 8. Computational overhead and estimation accuracy with respect to the order of the state function. The red line is the computational overhead and the bars are the mean square error.

First, we introduce an online model for $\mathbf{Q}_t$ as follows: let $\hat{x}_t$ be the estimated time, which is equivalent to $f_t(s)$ obtained from the *predictor* module. The estimation error covariance at time step t is the mean square error of the prior estimated values and the present estimated value. This is given by

$$\mathbf{Q}_t = \mathbb{E}[(\hat{x}_{t-1} - \hat{x}_t)^2], \tag{29}$$

$$= \frac{1}{t}\sum_{i=1}^{t}(\hat{x}_{i-1} - \hat{x}_i)^2. \tag{30}$$

Equation (30) is inefficient from the computational perspective, since the system must keep the entire history of estimates until time step t. To reduce the computational complexity of (30), we transform (30) in a recursive form, as shown in (31):

$$\mathbf{Q}_t = \frac{t-1}{t}\mathbf{Q}_{t-1} + \frac{1}{t}(\hat{x}_{t-1} - \hat{x}_t)^2. \tag{31}$$

Then, $\mathbf{Q}_t$ becomes a function of $\mathbf{Q}_{t-1}$, $\hat{x}_t$, and $\hat{x}_{t-1}$; hence, its computational complexity can greatly be reduced.

Similarly, the measurement error covariance is also computed in a recursive manner as follows:

$$\mathbf{R}_t = \frac{t-1}{t}\mathbf{R}_{t-1} + \frac{1}{t}(\mathbf{z}_t - \hat{x}_t)^2. \tag{32}$$

Note that we can make $\mathbf{Q}_t$ and $\mathbf{R}_t$ more sensitive to the estimation and measurement errors for capturing the nonstationarity by setting a finite-size window as discussed in [20].

5.3 v/f Selector

The basic concept of DVFS is to dynamically adjust the v/f pair to optimize the performance by making use of the trade-off between the performance and energy consumption. The proposed PF discussed in the previous section provides the decoding time estimate for each frame. The v/f selector determines an optimal v/f pair for each frame based on the estimate $\hat{x}_t$. The deadline and v/f switching overhead are denoted by *DL* and *SO*, respectively. Then, the optimal frequency for decoding the tth frame with the minimum energy consumption, but without violating DL, can be chosen as follows, because the reciprocal of time is the frequency:

$$f = \frac{1}{min\{\hat{x}_t + SO, DL\}}. \tag{33}$$

Obviously, DVFS-support processors provide only a finite number of discrete frequencies, as shown in Table 2. Let the supported frequency set $\mathbf{F} = \{f_0, f_1, \dots, f_{n-1}\}$, where n is the number of v/f pairs. Also, F is an ascending ordered set and f_i denote the frequency of row i in the DVFS table. Then, we choose v/f level, f_i, such that $f_i < f \leq f_{i+1}$ in order not to violate *DL*. In other words, if the calculated f is between f_{i-1} and f_i, then the v/f selector takes one step more than the calculated f in order not to violate DL. For example, when the processor uses seven v/f levels, f is 421 MHz by (33), and then f_i must be set to f_4 in Table 2, since it is usual to specify the minimum voltage level to support the given frequency in the specifications.

5.4 Implementation

We implemented two video decoders (MPEG and H.264) equipped with the proposed PF-DVFS. In our implementation, we integrated our method with one of the well-known video decoders called *mplayer* [26]. Note that we take a first-order polynomial for the state function, since it provides reasonable accuracy with marginal computation overhead. We show the pseudocode for the video decoders in Fig. 9. More details are as follows:

1. Line 1: We first initialize the parameters of PF-DVFS such as the number of particles, $\mathbf{Q}_t$, and $\mathbf{R}_t$. The initial weight of particle i is denoted by w_0^i. We initially assign the same weight to all particles by assuming a uniform distribution.
2. Lines 2-27: At each time step t (or each frame), the loop from line 2 to line 27 is executed for frame decoding with PF-DVFS.
3. Line 3: The type and size of the current frame are obtained from its header.
4. Lines 4-9: We estimate the decoding time of the current frame by summing the weighted particles. For the first two frames, we do not perform the estimation due to the lack of any previous history. In this case, we make a conservative assumption to avoid missing deadlines by setting the estimation to its deadline.
5. Lines 10-12: The estimated decoding time is passed to the v/f selector. The v/f selector determines the optimal v/f pair and adjusts the processor's v/f as discussed in Section 5.3.
6. Line 13: Once the v/f of the processor is set, the processor decodes the current frame.
7. Lines 14-16: Calculate each element of the matrix in (22). We can calculate these elements in a recursive manner in (23-26).
8. Lines 17-19: Update the state function $f_t(\cdot)$ as discussed in Section 5.2.1.
9. Line 20: Update both the measurement error covariance and estimation error covariance.
10. Lines 21-24: The particles are distributed based on the state function and error covariance. Also, we recalculate and normalize their weights. The weights w_t^i are obtained by (5), which becomes

```
Algorithm II. PF-DVFS(Adaptive Particle-filter)
input    number of particles $N_s$
         previous particles $\mathbf{x}_{t-1}$ and their weights $\mathbf{w}_{t-1}$
output   current particles $\mathbf{x}_t$ and their weights $\mathbf{w}_t$
         estimated decoding time $\hat{x}_t$
1:  initialize the particles, weights, and coefficients.
2:  for each frame do
3:      get size $s_t$ and frame type;
4:      if frame count is smaller than 2 then
5:          $\hat{x}_t$ = deadline;
6:      else
8:          $\hat{x}_t = \sum_{i=1}^{N_s}(x_t^i \cdot w_t^i)$;
9:      end if
10:     pass the $\hat{x}_t$ to v/f selector;
11:     obtain the right v/f pair;
12:     adjust v/f pair;
13:     decode the current frame;
14:     compute the elements of the matrix:
15:         $s_t^2$, $s_t$, $s_t x_t$, $x_t$;
16:     calculate Eqs. (23)-(26);
17:     if frame count is larger than 2 then
18:         solve Eqs. (27)-(28);
19:     end if
20:     calculate the error covariance by Eqs. (30)-(32)
21:     for each particles do
22:         distribute particles, $\mathbf{x}_t$;
23:         evaluate $\mathbf{w}_t$ using $f_t(\cdot)$, $\mathbf{Q}_t$, and $\mathbf{R}_t$;
24:     end for
25:     if $\hat{N}_{eff}$ is smaller than $N_{th}$ then
26:         resample;
27:     end if
28: end for
```

Fig. 9. Adaptive particle filter.

$$w_t^i \propto w_{t-1}^i \mathcal{N}(\mathbf{z}_t; \mathbf{b}_t, \mathbf{S}_t) \tag{34}$$

due to (11). In (34), $\mathbf{b}_t$ and $\mathbf{S}_t$ can be calculated by (13) and (12), respectively.

11. Lines 25-27: If $\hat{N}_{eff}$ is less than N_{th}, the resampling process is invoked to avoid the degeneracy problem. Unlike the generic PF, the reduced number of resampling also enables the proposed algorithm to avoid recalculating $\hat{N}_{eff}$ for every frame, since calculating $\hat{N}_{eff}$ is essential but large computational overhead by (6). The proposed algorithm computes it every 20 frames, which incurs only about 0.1 percent accuracy degradation according to our experiments.

6 Experimental Results

6.1 Experimental Setting

We conducted four sets of experiments to compare the performance of our methodology with that of the existing filter-based methods, such as WMA, PID, KF [20], and linear model,[1] including our own for MPEG and H.264 video clips. In the first set of experiments, we performed a series of experiments to tune the number of particles in our method. In the second set of experiments, we examined

1. We improved the linear model proposed in [6] such that it could perform online estimation as discussed in Section 5.

TABLE 1
Video Clips Used for Simulation

clips	resolution	# of frame	fps	MPEG				H.264			
				MAD	I_{MAD}	P_{MAD}	B_{MAD}	MAD	I_{MAD}	P_{MAD}	B_{MAD}
video clips with small workload variation (unit : $\times 10^3$)											
cindy	160x120	362	23.98	0.143	0.124	0.132	0.153	7.108	7.044	7.080	4.272
bobo	199x144	678	30	0.487	0.608	0.392	0.528	4.487	4.371	4.212	3.315
joel	160x120	99	23.98	0.079	0.070	0.071	0.392	1.389	5.916	0.932	0.856
bigbend	160x120	171	25	0.080	0.082	0.048	0.083	1.432	2.834	1.281	0.584
video clips with large workload variation (unit : $\times 10^3$)											
RedsNightmare	320x240	1209	25	2.369	2.383	2.242	2.339	14.215	11.774	12.274	16.536
berger	240x180	307	24	1.395	0.311	0.287	1.551	7.113	7.749	6.996	4.849
sl9_fraga_lin	256x256	196	30	0.988	0.315	0.898	1.126	5.417	5.531	4.328	5.775
tue	240x208	406	30	1.804	2.187	2.295	1.527	11.825	9.592	10.019	13.916

how quickly and precisely each method could track the variation of the workload. In the third set of experiments, we tested how well each technique cooperates with DVFS in terms of the estimation accuracy and energy saving, while satisfying the real-time constraint. In the fourth set of experiments, we performed a sensitivity analysis of each method with respect to the number of v/f pairs.

For these sets of experiments, we validated our method by measuring the energy consumption of an ARM1176-based testbed (SMDK6410 [25]) adopting the S3C6410 processor. This processor supports up to seven v/f pairs for DVFS and a maximum frequency of 800 MHz in Table 2. We extended a popular video player called *mplayer* [26] to support DVFS, while it is running on the evaluation board, to compare the performance of the aforementioned DVFS methods from various aspects including the energy saving.

We used eight video clips in all of the experiments. The clips are classified into two groups according to their mean absolute deviation (MAD), which is defined as the difference in the standard deviations of the absolute decoding times between two consecutive frames. MAD is more appropriate than the standard deviation of the frame decoding times[2] to quantify the nonstationarity of video clips, since it even considers the order of the frames to be decoded. Table 1 summarizes the characteristics of the video clips used in our experiments.

6.2 Tuning the Number of Particles

We first performed a series of experiments to tune the number of particles in our method, since it is a highly critical parameter to obtain a good trade-off between the estimation accuracy and runtime overhead (RO).

Fig. 10 shows the estimation accuracy and runtime overhead of our method with respect to the number of particles when *mplayer* decodes an H.264 video clip, "RedsNightmare." The accuracy of the PF saturates with a few particles (bold line), as shown in Fig. 10. More precisely, the estimation accuracy of our method becomes stable with less than 10 particles. This implies that our method exploits the application characteristics (the relationship between the frame size and decoding time) well for tuning the parameters of the PF. On the other hand, the computational overhead is almost linearly proportional to the number of particles, as shown by the dotted line in Fig. 10. Based upon the results shown in Fig. 10, we used 10 particles to validate our method.

2. Most previous approaches used this metric for indicating the nonstationarity of video clips.

6.3 Performance Comparison of DVFS Methods

We next evaluated the quality of our method with only 10 particles by comparing it with other methods in terms of the estimation accuracy and convergence speed. Fig. 11a shows a comparison of the speed of convergence of the different methods for achieving reasonable estimation accuracy. The estimation accuracy was measured in terms of the mean square error by comparing the estimated and actual decoding times. In our experiments, the proposed method (PF) outperformed all of the other techniques in terms of the estimation accuracy and convergence speed, even though it utilizes only 10 particles in order to minimize the computational overhead. We also compared the performance of all of the methods for all of the tested video clips in terms of the MSE, as shown in Fig. 11b. The results confirm that the PF outperforms all of the other methods, particularly for the video clips with large variations.

To compare the performance of the proposed method with that of the other existing methods more thoroughly from the DVFS perspective, we borrowed four useful metrics from [20] as follows:

TABLE 2
DVFS Table

Row #		Voltage[V]	Frequency[MHz]
4 pairs	7 pairs		
0	0	1.00	222
1	1	1.05	266
	2	1.15	333
2	3	1.20	400
	4	1.25	533
	5	1.25	667
3	6	1.30	800

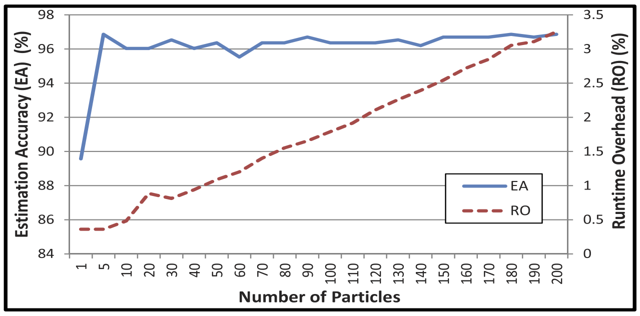
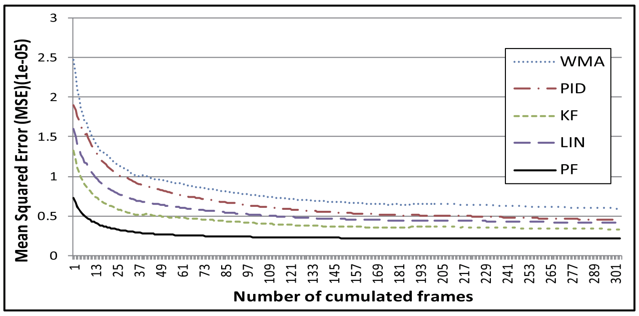
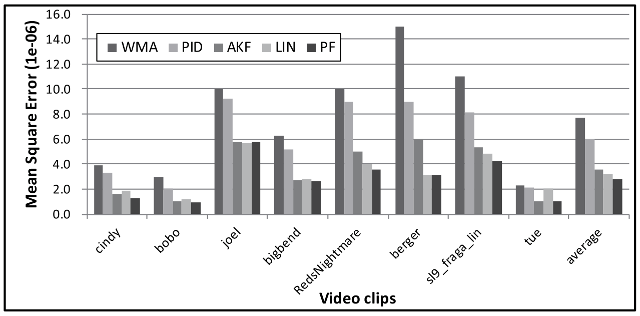


Fig. 10. Variation of the estimation accuracy and runtime overhead with the number of particles.



(a)



(b)

Fig. 11. Performance comparison. (a) Convergence speed of DVFS methods. (b) MSE comparison.

- The Decision Accuracy (**DA**) indicates how closely the DVFS with the target workload estimation method selects a v/f pair to the optimal v/f pair.
- The Hit Ratio (**HR**) is the ratio of the number of v/f pair selections that are identical to the selections of an ideal case to the total number of selections. We call the ideal case *oracle-DVFS*,which is possible only with offline analysis.
- The **EC** is the total energy consumed by a processor normalized to the energy consumption of the processor without DVFS.
- The Deadline Miss Ratio (**DMR**) is the ratio of the deadline-missing frames to the total number of frames.
- The Computation Overhead Ratio (**COR**) is the ratio of the increase in the execution time of the modified mplayer (with DVFS scheme) to the execution time of the original mplayer (without DVFS scheme). Note that the modified mplayer does not change its v/f pair, even though it performs the estimation. Thus, the runtime of modified mplayer includes the computing time for the estimation, but excludes the v/f pair switching time.

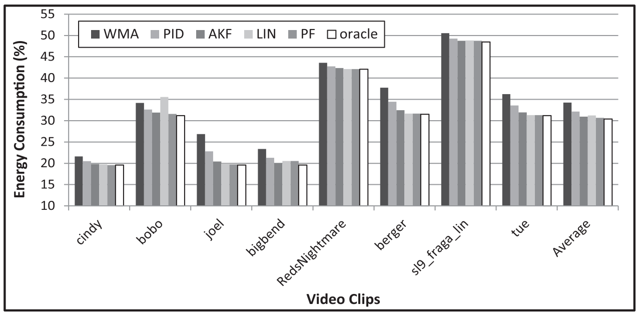
Based upon the above metrics, we compared the aforementioned DVFS methods, including our own, for both the MPEG and H.264 video clips, while setting the number of v/f pairs to four. The comparison results are summarized in Table 3, Figs. 12 and 13.

In Table 3, all of the compared methods showed impressive results for the MPEG video clips tested. More specifically, all of the methods achieved both DA and HR values higher than 90 percent, and there was not much difference in their efficiency. However, completely different results were obtained when they were applied to H.264, as shown in Table 3. In the case of H.264, all of the methods except for our approach showed a large degradation in both the DA and HR compared to the MPEG cases. For instance, the online linear model (LIN) showed a degradation in the HR of almost 50 percent. The filter-based methods showed less degradation compared to LIN, but their degradation ratio of HR was still more than 20 percent, while that of our method was only about 5 percent.

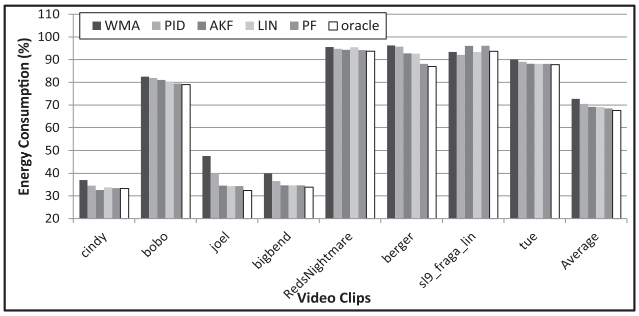
The different results in the case of MPEG and H.264 can be understood by comparing the MADs shown in Table 3, where the MAD of the H.264 video clips is much larger than that of the MPEG video clips. For instance, the MAD of the clip "cindy" in H.264 is about 50 times larger than that of the same clip encoded in MPEG. This is due to the more aggressive encoding scheme of H.264. In other words, the aggressive encoding scheme of H.264 increases the time-varying property of frame decoding, which results in a larger MAD. Even for the video clips with a large MAD, our method provides reasonable estimation accuracy in a stable manner, unlike the other methods, since it has an application-aware state function and error-correcting feedback mechanism. More specifically, the poor estimation accuracy

TABLE 3
Estimation Accuracy

clips	Decision Accuracy (DA, %)										Hit Ratio (HR, %)									
	MPEG					H.264					MPEG					H.264				
Small Var	WMA	PID	KF	LIN	PF	WMA	PID	KF	LIN	PF	WMA	PID	KF	LIN	PF	WMA	PID	KF	LIN	PF
cindy	97.52	98.90	99.72	99.45	100	67.68	69.52	72.37	98.07	99.72	97.52	98.90	99.72	99.72	99.72	27.90	26.80	36.74	97.24	99.45
bobo	94.30	96.07	97.59	99.70	99.70	87.02	88.25	89.43	91.51	93.64	90.57	91.75	93.67	99.85	99.85	69.32	68.88	69.91	75.95	81.45
joel	91.00	96.00	99.00	98.12	99.23	76.78	87.89	96.30	94.00	99.20	91.00	96.00	99.00	99.00	99.23	76.77	87.88	95.96	92.93	98.15
bigbend	94.31	96.07	97.59	99.70	99.70	87.26	92.14	95.46	96.18	99.45	95.31	97.92	99.48	99.48	99.48	82.72	86.91	90.05	95.28	99.00
Avg.	94.28	96.76	98.48	99.24	99.66	79.69	84.45	88.39	94.94	98.00	93.60	96.14	97.97	99.51	99.57	64.18	67.62	73.17	90.35	94.51
Large Var	WMA	PID	KF	LIN	PF	WMA	PID	KF	LIN	PF	WMA	PID	KF	LIN	PF	WMA	PID	KF	LIN	PF
RedsNight	93.89	93.89	95.90	99.81	99.81	94.90	95.56	96.53	85.07	97.82	86.12	83.64	88.18	99.83	99.83	89.58	89.58	90.74	56.99	94.18
berger	90.48	95.37	98.27	99.35	99.78	81.98	86.10	89.03	90.80	92.16	88.96	93.83	96.75	99.68	99.68	63.84	66.45	69.38	74.59	84.23
sl9_fraga	95.48	97.99	99.33	99.17	99.33	79.08	85.54	91.84	92.04	94.52	95.48	97.99	98.99	98.99	99.12	64.80	68.88	78.57	79.59	89.56
tue	92.63	95.90	97.62	99.51	99.72	89.82	93.27	95.40	87.63	97.32	91.15	93.61	94.35	99.51	99.51	83.25	86.45	87.93	66.75	96.13
Avg.	93.12	95.79	97.78	99.46	99.66	86.45	90.12	93.20	88.89	95.46	90.43	92.27	94.57	99.50	99.54	75.37	77.84	81.66	69.48	91.03
Total Avg.	93.70	96.27	98.13	99.35	99.66	83.07	87.28	90.80	91.91	96.73	92.01	94.21	96.27	99.51	99.55	69.77	72.73	77.41	79.92	92.77

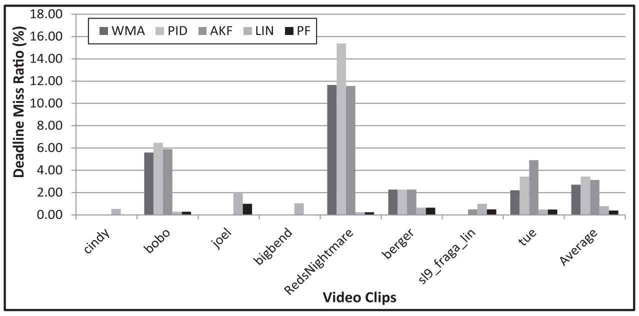


(a)

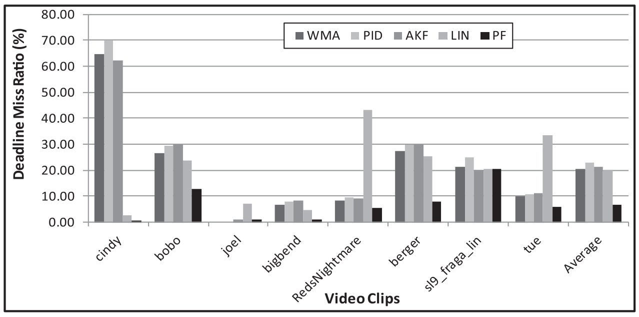


(b)

Fig. 12. Comparison of energy consumption. (a) MPEG video clips. (b) H.264 video clips.



(a)



(b)

Fig. 13. Comparison of deadline miss ratio. (a) MPEG video clips. (b) H.264 video clips.

of LIN and WMA is attributed to the lack of an error-correcting feedback mechanism, while the filter-based methods, except for ours, are ignorant of the application characteristics in their estimation.

The estimation accuracy of each method directly affects the quality of the DVFS. As shown in Fig. 12, all of the methods are comparable to the oracle case in terms of their energy saving, meaning that their energy efficiency is almost identical and close to the best efficiency that can be achieved. However, they show completely different efficiencies in maintaining the picture quality, as shown in Fig. 13. In Fig. 13a, all of the methods except our own and LIN show severe deadline misses for some of the video clips tested. In other words, they severely sacrifice the picture quality, even though their energy efficiency is almost identical to that of our method and LIN. The degradation of the picture quality in these methods becomes worse when they are applied to the H.264 video clips, due to the higher MADs. In this case, even the LIN model shows a large DMR for most of the video clips. However, our method shows a stable DMR which is 6.88 percent on average.

TABLE 4
Computation Overhead Ratio (COR, Percent)

	WMA	PID	AKF	LIN	PF
cindy	0.19	0.17	0.21	0.15	0.26
bobo	0.47	0.41	0.49	0.30	0.55
joel	0.24	0.21	0.26	0.16	0.29
bigbend	0.19	0.17	0.22	0.16	0.25
RedsNightmare	0.21	0.19	0.25	0.18	0.31
berger	0.35	0.30	0.38	0.27	0.43
sl9_fraga_lin	0.92	0.79	0.99	0.69	1.07
tue	0.35	0.30	0.38	0.21	0.38
average	0.37	0.32	0.40	0.27	0.44

Finally, we compared the COR for all of the methods in Table 4. Even though the PF shows superior performance in the other metrics, it also shows comparable performance to the other methods in terms of the COR.

To summarize, our method outperforms the other methods tested herein, especially for those video clips whose MADs are large. The superiority of our method is attributed to its application-aware state function and error-correcting feedback mechanism. Such estimation accuracy directly impacts on the quality of the DVFS and, hence, our method achieves an energy efficiency comparable to the oracle case, while sacrificing the picture quality only marginally, unlike in the case of the other methods.

6.4 Impact of v/f Levels

We also performed a sensitivity analysis of the DVFS methods with respect to the number of v/f pairs. For this purpose, we compared the energy saving and DMR of all of the methods when the numbers of v/f pairs are four and seven, respectively, and measured the HR, DA, EC, and DMR for each case. The results of the sensitivity analysis are shown in Table 5. For all of the metrics except for the energy consumption, all of the methods being compared show lower performance when the number of v/f pair increases to seven. In terms of the energy consumption, it is better to use more v/f pairs, since it is possible to utilize the slack time more efficiently according to (33). With respect to all the other metrics used, however, using more candidate pairs lowers the performances of the DVFS methods. Even in this situation, the performance degradation ratio of our method is much less than that of the others for all metrics except for the energy consumption.

TABLE 5
Comparison of Different Voltage/Frequency Levels

	Decision Accuracy (DA, %)										Hit Ratio(HR, %)									
	MPEG					H.264					MPEG					H.264				
v/f	WMA	PID	KF	LIN	PF	WMA	PID	KF	LIN	PF	WMA	PID	KF	LIN	PF	WMA	PID	KF	LIN	PF
4	93.70	96.27	98.13	99.35	99.66	83.07	87.28	90.80	91.91	96.73	92.01	94.21	96.27	99.51	99.55	69.77	72.73	77.41	79.92	92.77
7	93.45	96.21	97.99	98.83	99.39	82.99	86.99	90.61	86.89	96.53	87.26	89.22	91.16	94.92	98.52	54.90	55.23	60.96	59.84	90.30

	Energy Consumption (EC, %)										Deadline Miss Ratio (DMR, %)									
	MPEG					H.264					MPEG					H.264				
4	35.61	33.66	32.48	32.79	32.35	72.79	70.58	69.27	69.07	68.53	2.72	3.45	3.14	0.79	0.40	20.65	22.72	21.42	20.06	6.88
7	34.27	32.16	30.95	31.21	30.67	63.45	60.79	59.36	59.53	58.71	7.47	8.43	8.25	1.17	0.89	35.52	40.21	37.87	24.38	9.70

To summarize, all of the methods including our own provide higher energy efficiency with more v/f pairs. However, the picture quality becomes worse, due to the increased difficulty in selecting the optimal v/f pair. Nevertheless, our method is less sensitive to the number of v/f pairs, thanks to its highly accurate estimation capability.

7 Conclusion

In this paper, we proposed a novel DVFS technique for estimating the characteristics of nonstationary workloads. This technique exploits the particle filter, a sequential Monte Carlo method for inferring the posterior distribution in a Bayesian framework, and does not require any profiling or extensive training. We enhanced and customized the generic particle filter so that it can be used for estimating the execution time of the highly varying workloads encountered in DVFS applications by designing an application-aware state function and an error-correcting feedback mechanism.

The proposed technique was tested with real workloads obtained from MPEG and H.264 video clips in terms of various statistics. In particular, the proposed approach outperformed all of the DVFS techniques examined herein by 98.24 and 58.97 percent with respect to the accuracy and energy savings, respectively, with little overhead. Our results indicate that the proposed PF-based DVFS technique can be a highly effective tool for reducing the energy consumption in real-time embedded systems. As a final remark, we will extend this work to consider CPU-bound work and memory-bound work separately for further improving its effectiveness.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2012-0026822, 2011-0003559 and 2012-008475).

References

[1] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools.* Kluwer, 1997.
[2] G. Welch and G. Bishop, “An Introduction to the Kalman Filter,” technical report, Univ. of North Carolina at Chapel Hill, 1995.
[3] D. Shin, J. Kim, and S. Lee, “Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications,” *IEEE Design and Test of Computers,* vol. 18, no. 2, pp. 20-30, Mar. 2001.
[4] D. Shin and J. Kim, “A Profile-Based Energy-Efficient Intra-Task Voltage Scheduling Algorithm for Hard Real-Time Applications,” *Proc. Int'l Symp. Low Power Embedded Design,* pp. 271-274, 2001.
[5] J. Seo, T. Kim, and K.-S. Chung, “Profile-Based Optimal Intra-Task Voltage Scheduling for Hard Real-Time Applications,” *Proc. Design Automation Conf. (DAC),* pp. 87-92, 2004.
[6] A.C. Bavier, A.B. Montz, and L.L. Peterson, “Prediction MPEG Decoding Time,” *Proc. ACM SIGMETRICS Performance Evaluation Rev.,* pp. 131-140, June 1998.
[7] J. Pouwelse, K. Langendoen, I. Lagendijk, and H. Sips, “Power-Aware Video Decoding,” *Proc. Picture Coding Symp.,* pp. 303-306, 2001.
[8] S. Hong, S. Yoo, B. Bin, K.M. Choi, S.K. Eo, and T. Kim, “Dynamic Voltage Scaling of Supply and Body Bias Exploiting Software Runtime Distribution,” *Proc. Conf. Design, Automation and Test in Europe,* pp. 242-247, Mar. 2008.
[9] C. Xian, Y.-H. Lu, and Z. Li, “Dynamic Voltage Scaling for Multitasking Real-Time Systems with Uncertain Execution Time,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 27, no. 8, pp. 1467-1478, Aug. 2008.
[10] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, “A Tutorial on Particle Filters for On-Line Nonliear/Non-Gaussian Bayesian Tracking,” *IEEE Trans. Signal Processing,* vol. 50, no. 2, pp. 174-188, Feb. 2002.
[11] A. Doucet, S. Godsill, and C. Andreu, “On Sequential Monte Carlo Sampling Methods for Bayesian Filtering,” *Statistics and Computing* vol. 10, no. 3, pp. 197- 208, July 2000.
[12] F. Yao, A. Demers, and S. Shenker, “A Scheduling Model for Reduced CPU Energy,” *Proc. Ann. Symp. Foundation of Computer Science,* pp. 374-382, 1995.
[13] C. Im, H. Kim, and S. Ha, “Dynamic Voltage Scheduling Technique for Low-Power Multimedia Applications Using Buffers,” *Proc. Int'l Symp. Low Power Embedded Design,* pp. 34-39, 2001.
[14] R. Jejurika and R. Gupta, “Energy-Aware Task Scheduling with Task Synchronization for Embedded Real-Time Systems,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 25, no. 6, pp. 1024-1037, June 2006.
[15] Y. Tan, P. Malani, Q. Qiu, and Q. Wu, “Workload Prediction and Dynamic Voltage Scaling for MPEG Decoding,” *Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC),* 2006.
[16] M. Mesarina and Y. Turner, “Reduced Energy Decoding of MPEG Streams,” *Proc. Multimedia Computing and Networking,* 2002.
[17] K. Choi, W.-C. Cheng, and M. Pedram, “Frame-Based Dynamic Voltage Scaling for an MPEG Decoder,” *J. Low Power Electronics,* vol. 1, no. 1, pp. 44-51, Apr. 2005.
[18] A. Sinha and A.P. Chandrakasan, “Dynamic Voltage Scheduling Using Adaptive Filtering of Workload Traces,” *Proc. Int'l Conf. VLSI Design,* pp. 221-226, 2001.
[19] Y. Gu and S. Chakraborty, “Control Theory-Based DVS for Interactive 3D Games,” *Proc. Design Automation Conf.,* pp. 740-745, 2008.
[20] S.-Y. Bang, K. Bang, S. Yoon, and E.-Y. Li, “Run-Time Adaptive Workload Estimation for Dynamic Voltage Scaling,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 28, no. 9, pp. 1334-1347, Sept. 2009.
[21] J. Kim, S. Yoo, and C. Kyung, “Program Phase-Aware Dynamic Voltage Scaling under Variable Computational Workload and Memory Stall Environment,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 30, no. 1, pp. 110-123, Jan. 2011.
[22] K. Choi, R. Soma, and M. Pedram, “Dynamic Voltage and Frequency Scaling Based on Workload Decomposition,” *Proc. Int'l Symp. Low Power Electronic Devices,* pp. 174-179, 2004.
[23] K. Choi, R. Soma, and M. Pedram, “Off-Chip Latency-Driven Dynamic Voltage and Frequency Scaling for an MPEG Decoding,” *Proc. Design Automation Conf.,* pp. 544-549, 2004.

[24] R. van der Merwe, A. Doucet, N. de Freitas, and E. Wan, "The Unscented Particle Filter," Technical Report CUED/F-INFENG/TR 380, Cambridge Univ. ENg. Dept., 2000.
[25] SMDK6410, http://meritech.co.kr/eng/products/product_view.php?num=23, 2011.
[26] mplayer, http://www.mplayerhq.hu/design7/news.html, 2011.
[27] I. Richardson, "What Is H.264/AVC?" http://www.vcodex.com/h264/html, 2005.
[28] A. Doucet, N. de Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice.* Springer, 2001.
[29] A. Kong, J.S. Liu, and W.H. Wong, "Sequential Imputations and Bayesian Missing Data Problems," *J. Am. Statistical Assoc.*, vol. 89, no. 425, pp. 278-288, 1994.

Jae-Beom Lee received the BS and MS degrees in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2008 and 2010, respectively. He is currently working toward the PhD degree at the School of Electrical and Electronic Engineering, Yonsei University. His research interest includes system-level low-power design.

Myoung-Jin Kim received the BS degree in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2009. He is currently working toward the MS degree at the School of Electrical and Electronic Engineering, Yonsei University. His research interest includes system-level low-power design.

Sungroh Yoon (S'99-M'06-SM'11) received the BS degree in electrical engineering from Seoul National University, Korea, in 1996 and the MS and PhD degrees in electrical engineering from Stanford University, in 2002 and 2006, respectively. From 2006 to 2007, he was with Intel Corporation, Santa Clara. Previously, he held research positions at Stanford University and Synopsys, Inc., Mountain View. From 2007 to 2012, he was an assistant professor with the School of Electrical Engineering, Korea University, Seoul, Korea. Dr. Yoon is currently an assistant professor with the Department of Electrical and Computer Engineering at Seoul National University, Seoul, Korea. His research interests include biomedical computation and embedded systems. He is a senior member of the IEEE.

Eui-Young Chung received the BS and MS degrees in electronics and computer engineering from Korea University, Seoul, Korea, in 1988 and 1990, respectively, and the PhD degree in electrical engineering from Stanford University, California, in 2002. From 1990 to 2005, he was a principal engineer with SoC R&D Center, Samsung Electronics, Yongin, Korea. He is currently a professor with the School of Electrical and Electronics Engineering, Yonsei University, Seoul, Korea. His research interests include system architecture, biocomputing, and VLSI design, including all aspects of computer-aided design with the special emphasis on low-power applications and flash memory applications. He is a member of the IEEE.