Hindawi Publishing Corporation
Mobile Information Systems
Volume 2016, Article ID 1042525, 10 pages
http://dx.doi.org/10.1155/2016/1042525

Research Article

A Dynamic Programming Solution for Energy-Optimal Video Playback on Mobile Devices

Minseok Song and Jinhan Park

School of Computer and Information Engineering, Inha University, Incheon 22212, Republic of Korea

Correspondence should be addressed to Minseok Song; mssong@inha.ac.kr

Received 28 December 2015; Revised 8 April 2016; Accepted 11 April 2016

Academic Editor: Wenyao Xu

Due to the development of mobile technology and wide availability of smartphones, the Internet of Things (IoT) starts to handle high volumes of video data to facilitate multimedia-based services, which requires energy-efficient video playback. In video playback, frames have to be decoded and rendered at high playback rate, increasing the computation cost on the CPU. To save the CPU power, dynamic voltage and frequency scaling (DVFS) dynamically adjusts the operating voltage of the processor along with frequency, in which appropriate selection of frequency on power could achieve a balance between performance and power. We present a decoding model that allows buffering frames to let the CPU run at low frequency and then propose an algorithm that determines the CPU frequency needed to decode each frame in a video, with the aim of minimizing power consumption while meeting buffer size and deadline constraints, using a dynamic programming technique. We finally extend this algorithm to optimize CPU frequencies over a short sequence of frames, producing a practical method of reducing the energy required for video decoding. Experimental results show a system-wide reduction in energy of 27%, compared with a processor running at full speed.

1. Introduction

The Internet of Things (IoT) allows physical objects to interact and cooperate with one another by exchanging data, and multimedia-related services based on the IoT are now gaining popularity in various applications areas [1]. For example, users of home security systems now see images from cameras on a smartphone, and telemedicine systems allow doctors to monitor a patient's health using video communication.

To support multimedia applications within the IoT, the characteristics of video need to be considered carefully. For example, the amount of data involved requires the use of compression techniques for codecs, but encoding and decoding processes are computationally intensive. Video transmission is a real-time process, which requires continuously periodic decoding to avoid distorted playback. Most importantly, mobile IoT devices have a limited energy budget, making the energy requirements of video transmission an important issue.

An effective way of reducing CPU power consumption is to use a dynamic voltage and frequency scaling (DVFS) technique, which adjusts the operating voltage and frequency of the processor [2–4]. Because the energy dissipated by the CPU scales quadratically with the supply voltage, reducing the voltage saves a lot of energy but also slows program execution, so that an appropriate compromise is always required.

In video playback, frames have to be decoded and rendered at playback rate to avoid a loss of quality. For example, to play a video at 25 frames per second, a frame must be decoded every 40 ms. This decoding process needs to finish within this period, but workload imposed by each frame varies significantly with video content [5–10].

In most previous work on the application of DVFS to videos, the lowest frequency that satisfies the deadline of the decoding time is chosen to reduce power consumption [11], but more energy can be saved by introducing flexibility in timing, by means of buffering techniques: if several frames are decoded in advance, the CPU can operate at lower frequencies on average, but buffering comes with its own costs [12]. Therefore, power saving is only effective with an appropriate frequency selection method subject to buffer constraints, but previous work took no account of this issue.

We propose a new scheme that determines the CPU frequency needed to decode each frame, which minimizes energy consumption while avoiding buffer overrun. We start by developing a video playback and energy model, formulate the energy optimization problem, and go on to use a dynamic programming technique to determine a sequence of frequencies. We finally present experimental results based on measurement of smartphone energy consumption and decoding times.

The rest of this paper is organized as follows. We present related work in Section 2 and the system model in Section 3. We formulate an optimization problem in Section 4, propose a new frequency selection algorithm in Section 5, and extend it in Section 6. We assess our scheme in Section 7 and finally conclude the paper in Section 8.

2. Related Work

CPU power management has been the subject of a lot of research, and most of the resulting techniques involve either dynamic power management (DPM) or DVFS. DPM puts an idle CPU into sleep mode [13], whereas DVFS reduces the voltage and frequency of an active CPU [2, 4]. DPM is not generally suitable for real-time applications that run continuously, because the idle intervals are too short to allow the CPU to enter sleep mode [8]. Therefore, we only review previous works about DVFS only in this section.

DVFS techniques can be classified into interval-based and task-based algorithms [7, 14]. Interval-based schemes monitor the CPU load at intervals and respond by changing the CPU frequency and voltage. A representative scheme is the Linux Ondemand governor, which adjusts frequency periodically based on CPU utilization in the preceding interval [15]. Another scheme is LongRun [16] which varies the frequency to suit the measured utilization. These methods are typically easy to implement but can make inaccurate predictions based on the assumption that loads are similar to recent loads [14].

Task-based schemes can overcome this problem to some extent by classifying tasks into several types to which different frequency selection policies are applied. Ayoub et al. [17] manage frequency and voltage to meet a performance target, expressed as a fraction of maximum system performance. Flautner and Mudge [18] propose a method that chooses a CPU frequency for each task based on its recent computational requirements. Seo et al. [14] present a frequency allocation method to reduce the average response time of tasks. However, all of these methods have been developed for general workloads and therefore may not be suitable for multimedia applications with real-time constraints.

DVFS techniques for real-time systems are generally integrated with real-time scheduling [2–4]. Based on the analysis of worst-case execution times, they select CPU frequencies that satisfy the real-time constraints; but tasks are often complete before their worst-case execution times, so several algorithms incorporate methods of reclaiming the unused time [2, 4]. The CPU starts each period running at a frequency which will meet the worst-case demands and the frequency is then reduced in response to the actual computation requirement.

Several groups have investigated DVFS techniques for video applications [6, 7], in which the key issue is to estimate the computational requirements of successive frames. Most of these techniques predict the workload required to decode a frame from the workloads incurred in decoding previous frames and adjust the CPU frequency. The accuracy of these schemes has been improved by feedback mechanisms, which take previous prediction errors into account [19].

It has been widely observed [5–10] that frame decoding times vary significantly. For example, some of the frames in an MPEG video can take ten times as long to decode as an average frame [20]. That makes it difficult to estimate the computational requirements of successive frames to meet their deadlines [5–7]. Several workload estimation techniques have been proposed for video applications [5–7, 11, 19], and they can be categorized [5] into methods which make use of the relationship between the amount of data in a frame and decoding time and methods which predict decoding times based on recent times and aim to correct prediction errors using a feedback mechanism.

A close relationship between frame size and decoding time has been widely observed [5, 6, 11], especially in videos encoded with MPEG-style compression, and this relationship allows decoding times to be predicted with reasonable confidence. For example, Liu et al. [6] established a linear relationship between frame size and decoding time and used it to predict decoding times, while Yang and Song [11] improved the accuracy of this approach by introducing a logarithmic relationship, and Bavier et al. [21] used it to predict decoding times. Lee et al. [5] introduced particle-filter techniques to further improve the accuracy of this approach for H.264 codecs.

Yuan et al. [8–10] proposed several DVFS techniques in which the CPU speed is adjusted on the basis of a statistical analysis of past workloads. Urunuela et al. [7] developed a history-based DVFS technique, but it is tailored to video kiosks rather than general video players. Choi et al. [22] adopted a hybrid approach in which different DVFS policies are applied depending on the characteristics of each frame. Im and Ha [12] presented DVFS techniques in which buffers were used to reclaim unused CPU time, and Huang et al. [20] introduced a method of predicting decoding times from offline analysis of frame characteristics.

Most of these techniques do not consider the characteristics of video playback, in which some deadline misses and frame skipping are acceptable. Kim et al. [23] presented a DVFS scheme specifically for scalable video coding (SVC) codecs, which makes use of temporal scalability. The scheme put forward by Yang and Song [11] acknowledges the effect of the ratio of deadline miss on energy consumption, but this paper does not provide a satisfactory solution that selects the appropriate frequency while minimizing energy consumption, nor does it examine how buffering affects power consumption.

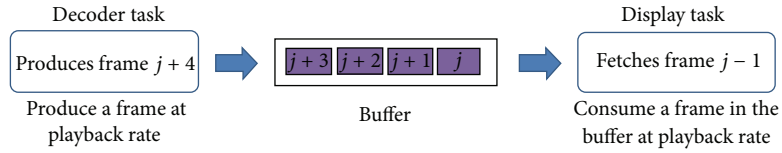

Figure 1: Video playback model.

3. Model

3.1. System Model. To support periodic nature of video playback, a video player decodes r frames per second, and so the decoding period of a frame, T^d, is $1/r$. The Notations explain important symbols used in this paper. Suppose that a CPU supports N^f frequency levels and that level k is the frequency f_k ($k = 1, \ldots, N^f$). If $k < m$, then $f_k < f_m$, so that f_{N^f} is the highest possible frequency. Let N^d be the number of frames decoded in a video. Let P_k^{active} and P_k^{idle}, respectively, be the active and idle power consumption of the system at frequency level k.

We will assume that the decoding time of each frame is known in advance: decoding times can be predicted by an offline analysis of the bitstream of a video [20] or by formulating a relationship between frame size and decoding time [5, 6, 11]. This decoding time information can be inserted into the header of a video [20], and we assume that these frames are available to our frequency selection algorithm. Specifically, $d_{j,k}$ is the decoding time of frame j at frequency level k.

3.2. Video Playback Model. Frame-level DVFS is appropriate for a media player [5–7, 11], which then selects the frequency which best matches the CPU workload imposed by the current frame, before that frame is decoded. The CPU does not change its frequency until the frame has been decoded.

Figure 1 shows our video playback model. The decoding task produces frames at playback rate and passes them to a buffer which stores frames for consumption by a display task, which fetches frames at playback rate. If there was no buffer, then only one frame can be handled by the display task, so the decoder enters sleep state until the frame is consumed by the display task. However, if a number of frames can be stored in the buffer, then decoding can run late, allowing lower frequencies to be selected, but this flexibility is limited by the size of buffer. For example, suppose that the buffer can accommodate N^b frames. If there are already N^b frames in the buffer, then decoding of a new frame must be delayed until the next decoded frame has gone to the display task. For example, consider Figure 1, where $N^b = 4$. If the buffer already contains 4 frames, then the decoder enters sleep state until frame j has been consumed by the display task.

To explain how this buffering technique can decrease CPU power consumption, consider a CPU with 4 frequency levels of 0.8 GHz, 1.2 GHz 1.6 GHz, and 1.8 GHz. We assume that $T^d = 40$ ms and that the process of a frame requires 36 ms at level 4, 40.5 ms at level 3, 54 ms at level 2, and 81 ms at level 1. If there was no buffer, then frequency level 4 must be chosen for every frame to keep the decoding time within 40 ms, as shown in Figure 2(a). However, if there is a buffer, which contains frames decoded when playback starts, then frequency level 1 can be selected for the first three frames, level 2 for the next two frames, and level 3 for the final frame as shown in Figure 2(b), without violating deadlines.

4. Problem Formulation

We formulate an optimization problem with a solution which will minimize energy consumption subject to the constraints of buffer size and decoding deadlines. Frame j ($j = 1, \ldots, N^d$) must be decoded before its deadline T_j^{end}, which is jT^d. At T_j^{end}, frame j leaves the buffer to be displayed on the screen. Let S_j be the frequency level selected for decoding frame j; and let T_j^{start} be the earliest possible time at which the decoding of frame j can start. Because the buffer can contain N^b frames, the decoding of frame j can start at $T_{j-N^b}^{\text{end}}$ (i.e., $(j-N^b)T^d$), at which frame $j-N^b$ can be removed from the buffer and displayed, allowing a new frame to be decoded and stored in the buffer. Thus, T_j^{start} can be expressed as follows:

$$T_j^{\text{start}} = \begin{cases} 0 & j = 1, \ldots, N^b \\ \left(j - N^b\right) T^d & j = N^b + 1, \ldots, N^d. \end{cases} \quad (1)$$

Let $T_{j,k}^{\text{over}}$ ($j = 1, \ldots, N^d - 1$) be the length of time by which the decoding of frame j would overrun if frequency level k is chosen, relative to the start time of the next frame T_{j+1}^{start}. The value of $T_{0,k}^{\text{over}}$ $\forall k$ is initialized to 0. This overrun can be expressed as follows:

$$T_{j,k}^{\text{over}} = \max\left(0, d_{j,k} + T_{j-1,S_{j-1}}^{\text{over}} + T_j^{\text{start}} - T_{j+1}^{\text{start}}\right). \quad (2)$$

If the decoding of frame j indeed finishes after T_{j+1}^{start}, then $T_{j,k}^{\text{over}}$ is the time difference between the actual time at which the decoding of frame j finishes (i.e., $d_{j,k} + T_{j-1,S_{j-1}}^{\text{over}} + T_j^{\text{start}}$) and T_{j+1}^{start}, when frequency level k is chosen for frame j. Conversely, if time remains after the decoding of frame j, the CPU enters its idle state and remains in this state until T_{j+1}^{start}, when $T_{j,k}^{\text{over}}$ is set to 0.

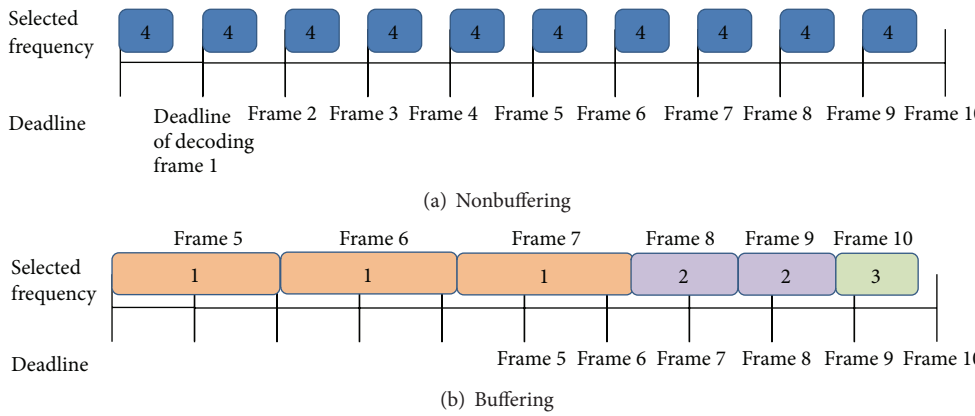

Figure 2: Comparison between buffering and nonbuffering.

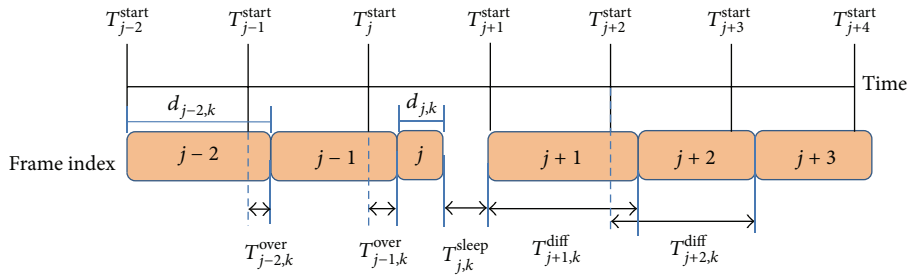

Figure 3: Sequence of frames showing the relationship between $T^{over}_{j,k}$, $T^{sleep}_{j,k}$, and $T^{diff}_{j,k}$.

If $T^{over}_{j,k} = 0$, then the CPU stays in its idle state for the length of $T^{sleep}_{j,k}$, which can be expressed as $T^{start}_{j+1} - (T^{start}_{j} + d_{j,k} + T^{over}_{j-1,S_j})$; otherwise, $T^{sleep}_{j,k}$ is set to 0. The determination of $T^{sleep}_{j,k}$ can thus be summarized as follows:

$$T^{sleep}_{j,k} = \begin{cases} T^{start}_{j+1} - T^{start}_{j} - d_{j,k} - T^{over}_{j-1,S_{j-1}} & \text{if } T^{over}_{j,k} = 0 \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

The energy consumed during the period of $d_{j,k} + T^{sleep}_{j,k}$ while frame j is decoded at frequency level k is written as $E_{j,k}$, which can be expressed as follows:

$$E_{j,k} = d_{j,k} P^{active}_{k} + T^{sleep}_{j,k} P^{idle}_{k}. \quad (4)$$

At this point, we must introduce a further variable $T^{diff}_{j,k}$, which is the difference between the actual time at which the decoding of frame j finishes ($d_{j,k} + T^{over}_{j-1,S_{j-1}} + T^{start}_{j}$) and T^{start}_{j}, and this difference can be expressed as follows:

$$T^{diff}_{j,k} = d_{j,k} + T^{over}_{j-1,S_{j-1}}. \quad (5)$$

Figure 3 shows the relationship between $T^{over}_{j,k}$, $T^{sleep}_{j,k}$, and $T^{diff}_{j,k}$ in a short sequence of frames.

The decoding of frame j must start after T^{start}_{j} and finish before T^{end}_{j}. We can express this period for each frame j ($j = 1, \ldots, N^d$) as T^{round}_{j}, so that $T^{round}_{j} = T^{end}_{j} - T^{start}_{j}$. Each frame j must be decoded before its deadline T^{end}_{j}, so $T^{diff}_{j,k} \leq T^{round}_{j}$. Our frequency selection policy has to minimize the total energy consumption $\sum_{j=1}^{N^d} E_{j,S_j}$. We can now formulate this frequency selection problem that determines S_j ($j = 1, \ldots, N^d$) as follows:

$$\begin{aligned} &\text{Minimize} \quad \sum_{j=1}^{N^d} E_{j,S_j} \\ &\text{Subject to} \quad T^{diff}_{j,S_j} \leq T^{round}_{j}, \quad \forall j,\ j = 1, \ldots, N^d. \end{aligned} \quad (6)$$

5. Frequency Allocation Algorithm

5.1. Algorithm Concept. We now propose an algorithm to solve the problem using a dynamic programming technique. We will use a resolution of 1 ms for time values such as $T^{diff}_{j,k}$. Let $V^{energy}_{j,u}$ be the minimum amount of energy when T^{diff}_{j,S_j} is u milliseconds and frame j is decoded ($j = 1, \ldots, N^d$ and $u = 1, \ldots, T^{round}_{j}$). Let $F_{j,u}$ be the frequency level required to achieve the energy consumption of $V^{energy}_{j,u}$; further, let $V^{over}_{j,u}$ be the corresponding value of T^{over}_{j,S_j} and $V^{diff}_{j,u}$ the value of T^{diff}_{j-1,S_j}.

Table 1: Table for the value of $V_{j,u}^{\text{energy}}$ against $T_{j,k}^{\text{diff}}$.

Frame number	$T_{j,k}^{\text{diff}}$ values				
	1	2	⋯	$T_j^{\text{round}} - 1$	T_j^{round}
1	$V_{1,1}^{\text{energy}}$	$V_{1,2}^{\text{energy}}$	⋯	$V_{1,T_1^{\text{round}}}^{\text{energy}}$	$V_{1,T_1^{\text{round}}}^{\text{energy}}$
2	$V_{2,1}^{\text{energy}}$	$V_{2,2}^{\text{energy}}$	⋯	$V_{2,T_2^{\text{round}}}^{\text{energy}}$	$V_{2,T_2^{\text{round}}}^{\text{energy}}$
⋮	⋮	⋮	⋯	⋮	⋮
N^d	$V_{N^d,1}^{\text{energy}}$	$V_{N^d,2}^{\text{energy}}$	⋯	$V_{N^d,T_{N^d-1}^{\text{round}}}^{\text{energy}}$	$V_{N^d,T_{N^d}^{\text{round}}}^{\text{energy}}$

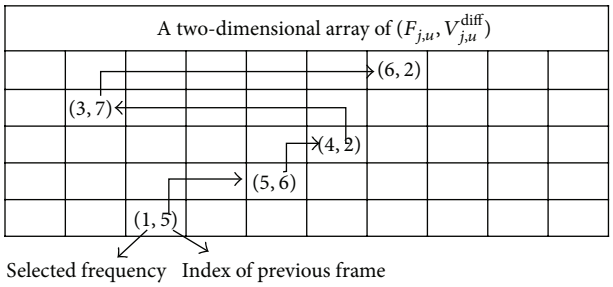

Figure 4: Process of backtracking.

The main idea of the dynamic programming is to construct a table of the optimal energy $V_{j,u}^{\text{energy}}$ for each frame j when $T_{j,k}^{\text{diff}} = u$ ($u = 1, \ldots, T_j^{\text{round}}$), as described in Table 1, where the minimum value in the final row, $\min_{u=1,\ldots,T_{N^d}^{\text{round}}} V_{N^d,u}^{\text{energy}}$, represents the amount of energy consumed by the optimal frequency allocation. For this purpose, we first initialize the values of $V_{1,u}^{\text{energy}}$ and then develop the recurrence relationship between consecutive frames so as to find all the values of $V_{j,u}^{\text{energy}}$ in the table.

We also maintain a two-dimensional array of tuples $(F_{j,u}, V_{j,u}^{\text{diff}})$ which leads to the minimum energy of $V_{j,u}^{\text{energy}}$ as illustrated in Figure 4. Using this array, a backtracking phase starts from frame N^d to frame 1 to select frequency of every frame. For example, Figure 4 shows an array of these tuples when $N^d = 5$ and $T_j^{\text{round}} = 10$. Suppose that the third column in the last row has the minimum energy value. Because $V_{j,u}^{\text{diff}}$ points to the column index of the previous frame, a sequence of frequencies can be selected as follows: (6, 3, 4, 5, 1). Likewise, our dynamic programming algorithm has three phases: (1) initialization, (2) establishment of recurrence relation, and (3) backtracking.

5.2. Initialization. For initialization, consider the following:

(1) $V_{j,u}^{\text{energy}}$, $V_{j,u}^{\text{over}}$, $V_{j,u}^{\text{diff}}$, and $F_{j,u}$ are all initialized to ∞ ($\forall j, u$, $j = 1, \ldots, N^d$ and $u = 1, \ldots, T_j^{\text{round}}$).

(2) $\forall k$, where $d_{1,k} \le T_1^{\text{end}}$, the values of $T_{1,k}^{\text{over}}$, $T_{1,k}^{\text{sleep}}$, and $E_{1,k}$ are calculated from (2), (3), and (4), respectively. Next, $V_{1,d_{1,k}}^{\text{over}}$ is replaced with $T_{1,k}^{\text{over}}$; then, $V_{1,d_{1,k}}^{\text{energy}}$ is updated to $E_{1,k}$, and $F_{1,d_{1,k}}$ is replaced with frequency level k.

5.3. Establishment of Recurrence Relation. During the recurrence establishment phase, $V_{j,u}^{\text{energy}}$, $F_{j,u}$, $V_{j,u}^{\text{diff}}$, and $V_{j,u}^{\text{over}}$ ($j = 2, \ldots, N^d$ and $u = 1, \ldots, T_j^{\text{round}}$) are updated as follows:

(1) For each value of j and u ($j = 2, \ldots, N^d$ and $u = 1, \ldots, T_j^{\text{round}}$), we maintain a two-dimensional array, $A_{j,u}(i,k)$ ($i = 1, \ldots, T_{j-1}^{\text{round}}$ and $k = 1, \ldots, N^f$). The following steps are repeated to find the value of $A_{j,u}(i,k)$, $\forall i, k$ if $V_{j-1,i}^{\text{energy}} \neq \infty$:

(a) Calculate the value of $T_{j,k}^{\text{over}}$ using (2), after replacing $T_{j-1,S_{j-1}}^{\text{over}}$ with $V_{j-1,i}^{\text{over}}$.

(b) Using the resulting value of $T_{j,k}^{\text{over}}$, calculate $T_{j,k}^{\text{sleep}}$ from (3).

(c) Using the resulting value of $T_{j,k}^{\text{sleep}}$, calculate $E_{j,k}$ from (4) and use this value of $E_{j,k}$ to update $A_{\{j,d_{j,k}+V_{j-1,i}^{\text{over}}\}}(i,k)$.

(2) $V_{j,u}^{\text{energy}}$, $F_{j,u}$, $V_{j,u}^{\text{diff}}$, and $V_{j,u}^{\text{over}}$ are updated as follows:

$$
\begin{aligned}
V_{j,u}^{\text{energy}} &= \min_{i=1,\ldots,T_{j-1}^{\text{round}},\ k=1,\ldots,N^f} \left(V_{j-1,i}^{\text{energy}} + A_{j,u}(i,k)\right),\\
F_{j,u} &= \arg \min_{k=1,\ldots,N^f} V_{j,u}^{\text{energy}},\\
V_{j,u}^{\text{diff}} &= \arg \min_{i=1,\ldots,T_{j-1}^{\text{round}}} V_{j,u}^{\text{energy}},\\
V_{j,u}^{\text{over}} &= \max\left(0, V_{V_{j,u}^{\text{diff}}} + d_{j,F_{j,u}} - T_{j+1}^{\text{start}}\right).
\end{aligned}
\tag{7}
$$

5.4. Backtracking. We find values of S_j using a backtracking technique as follows:

(1) j is initialized to N^d, and u is set to $\arg\min_{\{u=1,\ldots,T_{N^d}^{\text{round}}\}} V_{N^d,u}^{\text{energy}}$, so that $V_{N^d,u}^{\text{energy}}$ represents the amount of minimum energy consumption.

(2) While $j > 0$, the following procedures are repeated: (1) S_j is set to $F_{j,u}$, (2) $V_{j,u}^{\text{diff}}$ is substituted for u, and (3) j is decremented by 1.

Pseudocode for this frequency selection algorithm (FSA) is presented as Algorithm 1. If $T^{\max}$ is the maximum round length so that $T^{\max} = \max_{j=2,\ldots,N^d} T_j^{\text{round}}$, we can easily see from Algorithm 1 that the complexity of FSA is $O(N^d N^f T^{\max})$.

6. Algorithm Execution

If the frame decoding time is known in advance, then FSA can run without modification. For example, before playback, frequency allocation table during the entire playback can be obtained as a result of algorithm execution. However, since the algorithm complexity depends on the number of frames to be decoded, we divide the algorithm into iterations and limit the number of frames taken by the algorithm to

(1) Temporary variables: j, u, i and k;
(2) **for** $j = 1$ to N^d **do**
(3) **for** $u = 1$ to T_j^{round} **do**
(4) $V_{j,u}^{\text{energy}}$, $V_{j,u}^{\text{over}}$, $V_{j,u}^{\text{diff}}$ and $F_{j,u} \leftarrow \infty$;
(5) **end for**
(6) **end for**
(7) **for** $u = 1$ to T_j^{round} **do**
(8) **for** $k = 1$ to N^f **do**
(9) **if** $d_{1,k} \leq T_1^{\text{end}}$ and $u = d_{1,k}$ **then**
(10) Calculate $T_{1,k}^{\text{over}}$, $T_{1,k}^{\text{sleep}}$ and $E_{1,k}$ using (2), (3), and (4), respectively;
(11) $V_{1,u}^{\text{energy}} \leftarrow E_{1,k}$;
(12) $F_{1,u} \leftarrow k$;
(13) $V_{1,u}^{\text{over}} \leftarrow T_{1,k}^{\text{over}}$;
(14) **end if**
(15) **end for**
(16) **end for**
(17) **for** $j = 2$ to N^d **do**
(18) **for** $i = 1$ to T_{j-1}^{round} **do**
(19) **for** $k = 1$ to N^f **do**
(20) **if** $V_{j-1,i}^{\text{energy}} \neq \infty$ **then**
(21) Calculate the value of $T_{j,k}^{\text{over}}$ from (2) by replacing $T_{j-1,S_{j-1}}^{\text{over}}$ with $V_{j-1,i}^{\text{over}}$;
(22) $T_{j,k}^{\text{sleep}}$ is calculated from (3) by replacing $T_{j-1,S_{j-1}}^{\text{over}}$ with $V_{j-1,i}^{\text{over}}$;
(23) $E_{j,k}$ is calculated from (4), and $A_{\{j,d_{j,k}+V_{j-1,i}^{\text{over}}\}}(i,k)$ is updated using this value of $E_{j,k}$;
(24) **end if**
(25) **end for**
(26) **end for**
(27) $V_{j,u}^{\text{energy}} \leftarrow \min_{\{i=1,\ldots,T_{j-1}^{\text{round}} \text{ and } k=1,\ldots,N^f\}}(V_{j-1,i}^{\text{energy}} + A_{j,u}(i,k))$;
(28) $F_{j,u} \leftarrow \arg\min_{\{k=1,\ldots,N^f\}} V_{j,u}^{\text{energy}}$;
(29) $V_{j,u}^{\text{diff}} \leftarrow \arg\min_{\{i=1,\ldots,T_{j-1}^{\text{round}}\}} V_{j,u}^{\text{energy}}$;
(30) $V_{j,u}^{\text{over}} \leftarrow \max(0, V_{V_{j,u}^{\text{diff}}}^{\text{diff}} + d_{j,F_{j,u}} - T_{j+1}^{\text{start}})$;
(31) **end for**
(32) $j \leftarrow N^d$;
(33) $u \leftarrow \arg\min_{i=1,\ldots,T_{N^d}^{\text{round}}} V_{N^d,i}^{\text{energy}}$;
(34) **while** $j > 0$ **do**
(35) $S_j \leftarrow F_{j,u}$;
(36) $u \leftarrow V_{j,u}^{\text{diff}}$;
(37) $j \leftarrow j - 1$;
(38) **end while**

Algorithm 1: FSA (frequency selection algorithm).

N^l ($N^l < N^d$). Therefore, at the beginning of the mth iteration, the algorithm chooses the frequency for frames between $(m-1)N^l + 1$ and mN^l, which we call FSA-split, as shown in Algorithm 2.

FSA-split has the following characteristics in comparison with FSA:

(i) FSA-split determines the frequencies of frames between $(m-1)N^l + 1$ and mN^l.

(ii) An initialization part (lines between (9) and (22) in Algorithm 2) takes the length of overrun in the previous iteration ($V_{(m-1)N^l,I^{\text{prev}}}^{\text{over}}$) for the calculation of the parameter values.

Several methods were developed for decoding time estimation, most of which predict future decoding times based on recent measured times [5–7, 11, 19]. The decoding times of the frames in a certain GOP do not change a lot compared with those of its neighboring GOPs [11]. We can therefore predict the decoding times of the next GOP on the basis of those of the current GOP. For example, if N^l is set to the number of frames of a GOP, then the frequency allocation table can be established for the next GOP by passing predicted decoding times of the next GOP to the input parameters of the FSA-split.

7. Experimental Results

7.1. Setup. We performed simulations to evaluate our schemes using power data and timings obtained experimentally. The power consumption of a Samsung Nexus S smartphone (not just the CPU) was measured, and Table 2 shows its

(1) Temporary variables: j, u, i and k;
(2) Input parameter from the previous iteration ($m > 1$): I^{prev}
(3) $I^{\text{prev}} \leftarrow \arg\min_{i=1,\ldots,T^{\text{round}}_{(m-1)N^l}} V^{\text{energy}}_{(m-1)N^l,i}$;
(4) **for** $j = (m-1)N^l + 1$ to mN^l **do**
(5) **for** $u = 1$ to T^{round}_j **do**
(6) $V^{\text{energy}}_{j,u}$, $V^{\text{over}}_{j,u}$, $V^{\text{diff}}_{j,u}$ and $F_{j,u} \leftarrow \infty$;
(7) **end for**
(8) **end for**
(9) **for** $u = 1$ to T^{round}_j **do**
(10) **for** $k = 1$ to N^f **do**
(11) **if** $d_{(m-1)N^l+1,k} \le T^{\text{end}}_{(m-1)N^l+1}$ and $u = d_{(m-1)N^l+1,k}$ **then**
(12) **if** $m > 1$ **then**
(13) Calculate $T^{\text{over}}_{(m-1)N^l+1,k}$, $T^{\text{sleep}}_{(m-1)N^l+1,k}$ and $E_{(m-1)N^l+1,k}$ using (2), (3), and (4), respectively, by replacing $T^{\text{over}}_{j-1,S_{j-1}}$ with $V^{\text{over}}_{(m-1)N^l,I^{\text{prev}}}$;
(14) **else**
(15) Calculate $T^{\text{over}}_{(m-1)N^l+1,k}$, $T^{\text{sleep}}_{(m-1)N^l+1,k}$ and $E_{(m-1)N^l+1,k}$ using (2), (3), and (4), respectively, by replacing $T^{\text{over}}_{j-1,S_{j-1}}$ with 0;
(16) **end if**
(17) $V^{\text{energy}}_{(m-1)N^l+1,u} \leftarrow E_{(m-1)N^l+1,k}$;
(18) $F_{(m-1)N^l+1,u} \leftarrow k$;
(19) $V^{\text{over}}_{(m-1)N^l+1,u} \leftarrow T^{\text{over}}_{(m-1)N^l+1,k}$;
(20) **end if**
(21) **end for**
(22) **end for**
(23) **for** $j = (m-1)N^l + 2$ to mN^l **do**
(24) **for** $i = 1$ to T^{round}_{j-1} **do**
(25) **for** $k = 1$ to N^f **do**
(26) **if** $V^{\text{energy}}_{j-1,i} \neq \infty$ **then**
(27) Calculate the value of $T^{\text{over}}_{j,k}$ from (2) by replacing $T^{\text{over}}_{j-1,S_{j-1}}$ with $V^{\text{over}}_{j-1,i}$;
(28) $T^{\text{sleep}}_{j,k}$ is calculated from (3) by replacing $T^{\text{over}}_{j-1,S_{j-1}}$ with $V^{\text{over}}_{j-1,i}$;
(29) $E_{j,k}$ is calculated from (4), and $A_{\{j,d_{j,k}+V^{\text{over}}_{j-1,i}\}}(i,k)$ is updated using this value of $E_{j,k}$;
(30) **end if**
(31) **end for**
(32) **end for**
(33) $V^{\text{energy}}_{j,u} \leftarrow \min_{\{i=1,\ldots,T^{\text{round}}_{j-1} \text{ and } k=1,\ldots,N^f\}}(V^{\text{energy}}_{j-1,i} + A_{j,u}(i,k))$;
(34) $F_{j,u} \leftarrow \arg\min_{\{k=1,\ldots,N^f\}} V^{\text{energy}}_{j,u}$;
(35) $V^{\text{diff}}_{j,u} \leftarrow \arg\min_{\{i=1,\ldots,T^{\text{round}}_{j-1}\}} V^{\text{energy}}_{j,u}$;
(36) $V^{\text{over}}_{j,u} \leftarrow \max(0, V_{V^{\text{diff}}_{j,u}} + d_{j,F_{j,u}} - T^{\text{start}}_{j+1})$;
(37) **end for**
(38) $j \leftarrow mN^l$;
(39) $u \leftarrow \arg\min_{i=1,\ldots,T^{\text{round}}_{mN^l}} V^{\text{energy}}_{mN^l,i}$;
(40) **while** $j > (m-1)N^l$ **do**
(41) $S_j \leftarrow F_{j,u}$;
(42) $u \leftarrow V^{\text{diff}}_{(m-1)N^l+j,u}$;
(43) $j \leftarrow j - 1$;
(44) **end while**

Algorithm 2: FSA-split (frequency selection algorithm for the mth iteration).

active and idle power values. The time taken to decode video frames was also measured for the two videos in Table 3. We compared our scheme with two other algorithms as follows:

(1) HF always selects the highest frequency, which is equivalent to no DVFS.

Table 2: Measured power consumption against frequency of Samsung Nexus S phone.

Frequency (MHz)	1000	800	400	200	100
Active power (mW)	1324	1082	741	557	444
Idle power (mW)	545	527	503	471	420

Table 3: Video characteristics.

Type	Title	Resolution	Average bitrate	Playback time	GOP length
Animation	Ice Age 4	352 × 288	736 kb/s	5 minutes	12
Sports	Soccer	352 × 288	199 kb/s	5 minutes	12

Table 4: Relative energy consumption of FSA against a number of buffers.

Video type	Animation				Sports			
Number of buffers (N^b)	1	2	3	4	1	2	3	4
Energy used relative to HF	68.6%	67.1%	66.9%	66.9%	80%	77.8%	77.3%	77.2%
Energy used relative to LF	84.2%	82.3%	82.1%	82%	93.2%	90.6%	90.1%	89.9%

Table 5: Percentage of frames decoded at each frequency.

Video type	Animation					Sports				
Frequency (MHz)	100	200	400	800	1000	100	200	400	800	1000
LF	28.5%	3.7%	63.5%	4.3%	0%	0%	9.3%	90.4%	0.2%	0%
FSA ($N^b = 1$)	72%	0.8%	9.8%	14.8%	2.6%	4.8%	52.7%	31.1%	11.1%	0.3%
FSA ($N^b = 2$)	75.5%	0%	0.8%	19.9%	3.8%	24.7%	8.6%	27.2%	38.2%	1.3%
FSA ($N^b = 3$)	75.2%	0%	0.3%	19.7%	4.8%	28.1%	2.1%	21.8%	46.7%	1.3%
FSA ($N^b = 4$)	75.3%	0%	0.3%	20.0%	4.4%	28.9%	1.1%	18.4%	50.6%	1.1%

Table 6: Percentage energy difference between FSA and FSA-split against different values of N^l.

Video type	Animation					Sports				
N^l	12	24	48	96	192	12	24	48	96	192
$N^b = 1$	0.88%	0.48%	0.26%	0.14%	0.07%	0.22%	0.11%	0.05%	0.03%	0.02%
$N^b = 2$	1.26%	0.66%	0.33%	0.17%	0.09%	0.42%	0.21%	0.11%	0.06%	0.03%
$N^b = 3$	1.40%	0.70%	0.35%	0.18%	0.09%	0.49%	0.24%	0.12%	0.07%	0.04%
$N^b = 4$	1.47%	0.76%	0.38%	0.20%	0.10%	0.52%	0.24%	0.12%	0.06%	0.04%

(2) LF selects the lowest frequency level which will get each frame decoded in time. This method is a good heuristic, because CPU frequency can be expected to have a monotonic relationship with energy consumption [4–7, 11].

7.2. Efficacy of FSA. Table 4 shows how energy consumption depends on the number of frames that are buffered. We see that (1) FSA always shows the best performance, using 13% less energy than LF and 27% less energy than HF on average, and (2) increasing the size of the buffer saves more energy, but this amount of energy saved gradually tails off. In particular, even when $N^b = 1$ so that only one additional buffer is used, FSA uses 11% less energy than LF on average, suggesting that the buffer overhead of FSA is not high.

The results in Table 4 can be attributed to FSA's effective use of the slack times generated by storing decoded frames in the buffer, allowing the CPU to operate at lower frequencies. For example, Table 5 shows the average percentage of the frames in both video clips that are decoded at each frequency; FSA chooses lower frequencies than LF, decreasing energy consumption. FSA chooses the highest frequency (1000 MHz) more often than LF, which increases the idle time, allowing relatively lower frequencies to be chosen than LF. These results suggest that frequency selection has a great effect on energy consumption.

7.3. Efficacy of FSA-Split. To evaluate the efficacy of FSA-split, we examined how the values of N^l affect the energy consumption against different values of N^b as tabulated in Table 6. We see that (1) their energy difference is marginal, exhibiting 1.47% difference at maximum, even when N^l is set to 12 which is the GOP size; (2) increasing the value of N^l decreases the energy gap; and (3) increasing the buffer size increases the energy gap even though the difference is negligible. Although FSA exhibits slightly better performance than FSA-split, it takes all frame parameters for algorithm execution, requiring a lot of computation. These results suggest that FSA-split is a practical method of reducing the energy required for video decoding.

8. Conclusions

We have proposed a new frequency allocation scheme which minimizes energy consumption while avoiding buffer overrun, using a dynamic programming technique. This

scheme establishes recurrence relationship between consecutive frames to construct a table of the minimum energy values required to decode each frame and determines a sequence of frequencies required to decode every frame using a backtracking technique. It was extended to optimize CPU frequencies over a short sequence of frames, which gives a basis for energy-saving video decoding in practice.

Experimental results show that it uses 27% less energy than a processor at the highest frequency on average. In particular, it uses 13% less energy, compared to the widely used heuristic which chooses the lowest frequency to get each frame decoded in time. We believe that these results give a useful guideline for low-power video service by providing the minimum bound on power consumption required for video playback.

Notations

r: Playback rate of a video (fps)
N^f: Number of frequency levels supported by a CPU
T^d: Decoding period of a video
f_k: Frequency corresponding to the frequency level k
P_k^{active}: Active power at frequency level k
P_k^{idle}: Idle power at frequency level k
N^b: Number of frames that a display buffer can accommodate
N^d: Number of frames decoded in a video
$d_{j,k}$: Decoding time of frame j at frequency level k
T_j^{end}: Decoding deadline for frame j
T_j^{start}: Earliest possible start time for decoding frame j
T_j^{round}: $T_j^{\text{end}} - T_j^{\text{start}}$
$T_{j,k}^{\text{over}}$: $\max(0, d_{j,k} + T_{j-1,S_{j-1}}^{\text{over}} + T_j^{\text{start}} - T_{j+1}^{\text{start}})$
$T_{j,k}^{\text{sleep}}$: CPU sleep time when frequency level k is chosen for frame j
$E_{j,k}$: Energy consumed during the decoding period for frame j at frequency k
$A_{j,u}(i,k)$: Two-dimensional array for $E_{j,k}$ when $i = T_{j-1,S_j}^{\text{diff}}$, $u = T_{j,k}^{\text{diff}}$, and k is the frequency level chosen for decoding frame j
$T_{j,k}^{\text{diff}}$: Time between completion of decoding frame j and T_j^{start}
S_j: Frequency level selected for decoding frame j
$V_{j,u}^{\text{energy}}$: Minimum energy consumption in decoding frames 1 to j, when $T_{j,k}^{\text{diff}} = u$ ms
$F_{j,u}$: Frequency level selected for decoding frame j to achieve an energy of $V_{j,u}^{\text{energy}}$
$V_{j,u}^{\text{over}}$: Value of T_{j,S_j}^{over} to achieve an energy of $V_{j,u}^{\text{energy}}$
$V_{j,u}^{\text{diff}}$: Value of $T_{j-1,S_j}^{\text{diff}}$ to achieve an energy of $V_{j,u}^{\text{energy}}$
N^l: Number of frames for which frequencies are determined by FSA-split.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This research is supported by Inha University Research Grant.

References

[1] S. A. Alvi, B. Afzal, G. A. Shah, L. Atzor, and W. Mahmood, "Internet of multimedia things: vision and challenges," *Ad Hoc Networks*, vol. 33, pp. 87–111, 2015.

[2] H. Aydin, P. Mejia-Alvarez, D. Mosse, and R. Melhem, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, p. 95, London, UK, December 2001.

[3] M. Marinoni and G. Buttazzo, "Elastic DVS management in processors with discrete voltage/frequency modes," *IEEE Transactions on Industrial Informatics*, vol. 3, no. 1, pp. 51–62, 2007.

[4] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the ACM Symposium on Operating Systems Principles*, pp. 89–102, October 2001.

[5] J.-B. Lee, M.-J. Kim, S. Yoon, and E.-Y. Chung, "Application-support particle filter for dynamic voltage scaling of multimedia applications," *IEEE Transactions on Computers*, vol. 61, no. 9, pp. 1256–1269, 2012.

[6] X. Liu, P. Shenoy, and M. D. Corner, "Chameleon: application-level power management," *IEEE Transactions on Mobile Computing*, vol. 7, no. 8, pp. 995–1010, 2008.

[7] R. Urunuela, G. Muller, and J. L. Lawall, "Energy adaptation for multimedia information kiosks," in *Proceedings of the 6th ACM and IEEE International Conference on Embedded Software (EMSOFT '06)*, pp. 223–232, October 2006.

[8] W. Yuan and K. Nahrstedt, "Practical voltage scaling for mobile multimedia devices," in *Proceedings of the 12th Annual ACM International Conference on Multimedia (MULTIMEDIA '04)*, pp. 924–931, New York, NY, USA, October 2004.

[9] W. Yuan and K. Nahrstedt, "Energy-efficient CPU scheduling for multimedia applications," *ACM Transactions on Computer Systems*, vol. 24, no. 3, pp. 292–331, 2006.

[10] W. Yuan, K. Nahrstedt, S. V. Adve, D. L. Jones, and R. H. Kravets, "GRACE-1: cross-layer adaptation for multimedia quality and battery energy," *IEEE Transactions on Mobile Computing*, vol. 5, no. 7, pp. 799–815, 2006.

[11] A. Yang and M. Song, "Aggressive dynamic voltage scaling for energyaware video playback based on decoding time estimation," in *Proceedings of the ACM International Conference on Embedded Software*, pp. 1–9, Grenoble, France, October 2009.

[12] C. Im and S. Ha, "Dynamic voltage scaling for real-time multi-task scheduling using buffers," in *Proceedings of the ACM Conference on Languages, Compilers and Tools for Embedded Systems*, pp. 88–94, Washington, DC, USA, June 2004.

[13] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation (OSDI '94)*, article 2, USENIX Association, 1994.

[14] E. Seo, S. Park, J. Kim, and J. Lee, "TSB: a DVS algorithm with quick response for general purpose operating systems," *Journal of Systems Architecture*, vol. 54, no. 1-2, pp. 1–14, 2008.

[15] V. Pallipadi and A. Starikovskiy, "The ondemand governor: past, present, and future," in *Proceedings of the Linux Symposium*, pp. 223–238, Ottawa, Canada, July 2006.

[16] M. Fleischmann, "Longrun power management—dynamic power management for crusoe processors," Tech. Rep., Transmeta, 2001.

[17] R. Ayoub, U. Ogras, E. Gorbatov et al., "OS-level power minimization under tight performance constraints in general purpose systems," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '11)*, pp. 321–326, IEEE, Fukuoka, Japan, August 2011.

[18] K. Flautner and T. Mudge, "Vertigo: automatic performance-setting for linux," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, pp. 105–116, December 2002.

[19] Y. Gu and S. Chakraborty, "A hybrid DVS scheme for interactive 3D games," in *Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '08)*, pp. 3–12, St. Louis, Mo, USA, April 2008.

[20] Y. Huang, S. Chakraborty, and Y. Wang, "Using offline bitstream analysis for power-aware video decoding in portable devices," in *Proceedings of the 13th ACM International Conference on Multimedia (MM '05)*, pp. 299–302, Singapore, November 2005.

[21] A. Bavier, A. Montz, and L. Peterson, "Prediction MPEG decoding time," in *Proceedings of the ACM SIGMETRICS Conference*, pp. 131–140, Madison, Wis, USA, June 1998.

[22] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram, "Frame-based dynamic voltage and frequency scaling for a MPEG decoder," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD '02)*, pp. 732–737, ACM, November 2002.

[23] E. Kim, H. Jeong, J. Yang, and M. Song, "Balancing energy use against video quality in mobile devices," *IEEE Transactions on Consumer Electronics*, vol. 60, no. 3, pp. 517–524, 2014.