Multimedia Systems (2009) 15:113–124
DOI 10.1007/s00530-009-0153-5

REGULAR PAPER

An optimal speed control scheme supported by media servers for low-power multimedia applications

Wendong Huang · Ye Wang

Received: 1 December 2007 / Accepted: 16 February 2009 / Published online: 12 March 2009

Abstract In this paper, we present a new concept of dynamic voltage scaling (DVS) for low-power multimedia decoding in battery-powered mobile devices. Most existing DVS techniques are suboptimal in achieving energy efficiency while providing the guaranteed playback quality of service, which is mainly due to the inherent limitations of client-only approaches. To address this problem, in this paper, we investigate the possibility of media server supported DVS techniques with smoothing mechanisms. Towards this new direction, we propose a generic offline bitstream analysis framework and an optimal speed control algorithm which achieves the maximal energy savings among all feasible speed profiles for the given buffers. The proposed scheme enables us to compute the buffer sizes of feasibility condition, which are the theoretical lower bound of buffer size requirement for a given media clip. More importantly, our scheme facilitates practical applications from four aspects. First, it does not require feedback information on clients' configuration. This renders our scheme particularly suitable for broadcast or multicast applications. Second, the speed profile based on buffer sizes of feasibility condition can provide satisfactory energy efficiency. Third, the required buffer sizes are so small that they can be met by most mobile devices. Fourth, additional side information (i.e., speed profile) of the proposed scheme is negligible compared to the size of media content. These properties solve the diversity issue and feasibility issue of media server supported DVS schemes. Experimental results show that, in comparison with the representative existing techniques, our scheme improves the performance of DVS significantly.

Keywords Media server supported DVS · Smoothing · Offline bitstream analysis · Optimal speed control scheme

Communicated by Ralf Steinmetz.

W. Huang · Y. Wang (✉)
School of Computing, National University of Singapore,
Computing 1, 13 Computing Drive, Singapore 117417, Singapore
e-mail: wangye@comp.nus.edu.sg

W. Huang
e-mail: huangwd@comp.nus.edu.sg

1 Introduction

Low energy consumption is always a desired feature for battery-powered mobile devices such as mobile phones, PDAs and audio/video players. With the rapid growth of multimedia decoding applications running on these platforms, energy efficient methods optimized for such applications are becoming increasingly important.

Dynamic voltage scaling (DVS) is one of the most commonly used approaches to reduce the processor energy consumption, which adjusts the clock frequency and/or supply voltage level at run time while meeting the required quality of service (QoS). Video decoding applications, a popular application running on mobile devices, have large variations in workload, which proved to be a major challenge for DVS. When applied to video decoding, the performance of a DVS scheme can be evaluated in terms of its energy efficiency and associated playback QoS. Different DVS schemes provide different tradeoffs between these two requirements. In the class of hard real-time DVS schemes, the QoS is guaranteed at the cost of degradation of energy efficiency. This class of approaches performs the DVS operations using a global worst case execution time [1], or adaptive worst case execution times [2,3]. Thus the energy savings achieved by this class of approaches are limited since the large variations in workload of video decoding cannot be fully exploited. This leads to the class of soft real-time DVS schemes. As video decoding exhibits non-stationary workload characteristics, the conventional interval-based workload prediction methods result in unacceptably suboptimal solutions [4,5]. The effectiveness of DVS techniques largely depends on the capability of predicting the workload of the video decoding. To address this issue, three subclasses of approaches have been developed. The first subclass improves the prediction accuracy by incorporating frame parameters of the video bitstream into estimation, such as frame types [6], code sizes [7], etc, since it is shown that there is a strong correlation between workload and these parameters. The second subclass meets a certain percentage of frame deadlines based on the probability distribution of workload demands [8]. This subclass of work provides tunable tradeoffs between workload threshold and QoS. In the third subclass [9], the workload information is supplied by content providers in conjunction with the video clips, so workload prediction at the client site is not needed. Despite a lot of published work, workload prediction remains to be a challenge.

To alleviate the accuracy issue of the workload prediction, some work explored the possibility of avoiding the missed deadlines by buffering mechanisms. This concept can be traced back to [10], where processor speed is dynamically scaled based on the filling level of the input buffer to avoid its overflow or underflow. In recent years, buffer-based DVS techniques have been developed to compensate for the inaccuracy of the workload prediction [11], to average the workload of multiple frames [12,13], or to reduce the idle periods of the processor [14]. In general, buffer-based DVS approaches can achieve significant energy savings. According to our analysis, this is mainly because fluctuations of video decoding workload are smoothed out by buffers. As energy consumption of a processor is a convex function of its speed [15], energy consumption increases with the degree of fluctuation in the processor speed with the same average workload. This can be demonstrated by a simple example. Consider a decoding task with two frames: for case A, both frames require the same processor speed of 3, and for cases B and C, they require processor speeds of 2 and 4, 1 and 5, respectively. Thus cases A, B and C have the same average workload, but their fluctuation levels increase. We assume that the energy consumption of the processor is the square of its speed which is a convex function, we then have $E_A = 3^2 + 3^2 < E_B = 2^2 + 4^2 < E_C = 1^2 + 5^2$.

As video decoding exhibits large fluctuations in workload, smoothing becomes an effective method to reduce energy consumption. The attractiveness of smoothing stems from that energy consumption can be substantially reduced without sacrificing the playback quality [12]. For users, it offers an appealing compromise between prolonging the battery life and small latency (in our scheme, an averaged latency of less than 0.1 s will be introduced, which is negligible). Furthermore, our scheme can be implemented on an existing

II. PROBLEM FORMULATION

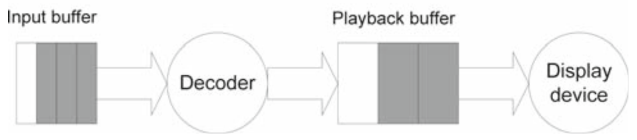

Fig. 1 Architecture of the multimedia processing system at the client site

video decoder since most video decoders have already made use of input and playback buffers to improve performance (refer to Fig. 1). These facts suggest that smoothing techniques are promising for energy efficient video decoding on mobile devices.

The effect of smoothing techniques has received little attention, and consequently existing buffer-based DVS techniques yield unsatisfactory performance in terms of smoothing. In these techniques, buffers provide two conflicting functions: (1) smoothing out fluctuations of the processor speed to reduce energy consumption; (2) avoiding missing deadline to guarantee QoS. Consequently, QoS requirements interfere with smoothing effects and their energy efficiency is degraded. Taking the algorithm proposed in [11] for example, the processor speed is scaled according to the filling level of buffer following control-theoretic principles. To avoid overflow and underflow of the buffer, sufficient marginal space of the buffer needs to be reserved during the speed control process. This shows that the buffer space is not fully exploited to achieve smoothing.

Motivated by this observation, we have the following important question: is it possible to address energy efficiency and QoS separately in a DVS scheme, let buffers focus on energy savings, and exploit an alternative mechanism for guaranteeing QoS? This question represents a novel insight into the DVS approach. When having accurate knowledge of workload and bit length of the decoding units, we can avoid missing deadlines through bitstream analysis, rather than the filling level of buffer. This can eliminate unnecessary speed scaling operations and fluctuations of the processor speed are further smoothed out.

This strategy, however, cannot be implemented by current techniques. In most existing DVS approaches, the client is solely responsible for the DVS operations and these client-only schemes have their inherent limitations. First, it is hard to obtain accurate workload information using prediction techniques in client-only schemes, which is clearly shown in our brief literature survey. Furthermore, due to the real-time requirement and limited computing resources, these client-only approaches can only afford computationally efficient but suboptimal solutions. On the other hand, these issues can be solved in the media server. The server can: (1) obtain the accurate workload estimation by simulation or measurement;

and (2) employ powerful computation resources to yield the globally optimal solution via offline video bitstream analysis. In addition, this method shifts the major computations relevant to energy efficiency from the client to the server. This simplifies the design of the client speed control scheme significantly. Based on these observations, in this paper, we investigate the possibility of a media server supported DVS which generates the speed profile for a given video bitstream at the server site offline by incorporating the accurate workload estimation and smoothing mechanisms. The resulting speed profile is then sent to clients together with the video content. At runtime, instead of performing the conventional DVS operations, the client reads the associated speed profile and scales the processor speed accordingly. To the best of our knowledge, the proposed approach has not been studied before.

The proposed scheme has a wide range of applications with pre-recorded media contents such as video on demand or simply download and playback media content. For practical applications, two critical issues need to be addressed. The first is the diversity issue where the clients are equipped with different processors or memory sizes. It gives rise to various video decoding workload and available buffer sizes for DVS. In comparison with buffer sizes, workload is less critical since: (1) for some video formats, such as MPEG-1 and MPEG-2, decoding workload on different processors can be translated by simple scaling computation [9]; (2) the different types of processors for mobile devices are very limited, for those formats where the scaling method cannot yield accurate estimation, we can generate a speed profile for each type of processors, based on the workload measurement on the target processor, and put all of these speed profiles into the side information. Since the size of a single speed profile is so small (details found in Table 4), the overhead associated with the total speed profiles are negligible. To address the problem of the different buffer sizes adopted by various clients, a possible solution is to generate different speed profiles for different groups of memory sizes. However, this is unnecessary in the case of our scheme. The second issue is the overhead associated with the speed profile. Since we will insert the speed profile as side information into the bitstream, it is desirable to make the size of the additional side information as small as possible. In the following we will show that our scheme has solved the above issues satisfactorily.

Besides the concept of media server supported DVS, the other contributions of this paper are as follows.

First, we have developed a bitstream analysis framework to address the inherent fluctuation of the decoding workload of a given media bitstream by manipulating the sizes and decoding workload of its media units, input and playback buffers of media decoders. As a generic framework, it can be used to solve various problems and provides analytical results.

Second, based on the preceding framework, we have proposed an algorithm to compute the optimal speed profile which achieves the minimal energy consumption among all feasible speed profiles with guaranteed QoS.

Third, in our scheme the buffer size requirement is determined by the characteristics of the given video clip through bitstream analysis. This scheme enables us to compute the buffer sizes of feasibility condition, which are the theoretical lower bound of the buffer size requirement to guarantee the playback QoS. Our scheme has following important properties:

- It does not require feedback information about the actually equipped buffers from clients. This will benefit the practical applications by simplifying the design of the whole system significantly.
- The energy consumption converges rapidly with increasing buffer size. As shown in Sect. 4.2, we achieve less than 2% (Table 3) additional energy reduction as we increase the buffer sizes from feasibility condition by 33–50%. This implies that we can compute a single speed profile based on the buffer sizes of the feasibility condition which yields satisfactory performance of energy efficiency.
- For a given bitstream, the buffer sizes of feasibility condition are so small (on average, 14.40 K bytes of input buffer and 453.48 K bytes of playback buffer, details found in Table 1) that they can be met by most mobile devices. This property satisfactorily solves the diversity issue of mobile devices. Meanwhile, the reduction of buffer requirements provides additional opportunities for energy savings. Through a memory controlling mechanism [16], the unused memory can be shut down or switched to an idle state to save energy. In such cases, the energy reduction is closely related to the reduction of buffer requirement. As memory operations are responsible for a significant portion of the overall energy consumption, the reduction of buffer requirements has an important contribution to the overall energy efficiency.
- The overhead of the speed profile is negligible compared to the video content size (details found in Table 4). Our scheme will keep the processor speed as a constant (details can be found in Sect. 3.3) until the speed must be changed due to the constraints of the buffers. Consequently, scaling operations of the processor speed are reduced significantly. This property allows us to update the speed only when it is changed, which is superior to the periodic sampling method of profile information used in existing schemes [9].

The paper is organized as follows. Section 2 formulates the problem. In Sect. 3, we present our solutions in three parts. In Sect. 4, we evaluate the proposed scheme. We conclude

Table 1 Configurations of decoding the six video clips

Clip name	Akiyo	Coastguard	Container	Hall	Highway	Walk
FI (Macroblock)	670	**1,320**	866	881	**1,249**	1,367
FP (Macroblock)	**1,052**	1,028	**1,035**	**1,029**	1,062	**1,401**
IB (KB)	12.87	15.97	12.76	12.54	15.37	16.90
PB (KB)	403.9	506.9	397.4	395.1	479.6	538.0
Delay (s)	0.088	0.11	0.075	0.076	0.10	0.12

FI and FP Feasibility condition for input buffer and playback buffer respectively, both of them measured in Macroblocks, the value in bold is used for both input buffer and playback buffer to estimate the other items, *IB and PB* input buffer size and playback buffer size, measured in Kbytes, both of them derived from the max(FI,FP), *Delay* introduced delay by buffering in s

the paper in Sect. 5. In Sect. 6, we prove the optimality of our speed profile algorithm.

2 Problem formulation

In this paper, we consider the following multimedia processing system architecture as shown in Fig. 1. The targeted client consists of: (1) an input buffer which is used to store the incoming compressed media stream before being processed; (2) a playback buffer which accommodates the decoded media data for display devices. Both the input and playback buffers have fixed capacities and work in a First In, First Out (FIFO) manner. For the sake of generality, we model the power–speed relationship of the processor as a convex function. The system-level view of the media bitstream throughout this paper is as follows: it is made up of a sequence of media objects. A media object can be a frame or a Macroblock. Before compression, all these objects have an identical bit length. However, the bit lengths of the media objects can be changed during the encoding process. Furthermore, the workload for decoding these media objects is also different. This model represents most video bitstreams generated by current compression techniques, such as MPEG video.

For a given media bitstream over period $[0, T]$, our proposed method will produce a speed profile $\pi = \{(\omega_1, t_1), \ldots, (\omega_n, t_n)\}$, with $t_0 = 0$, $t_n = T$, which means the processor speed is set to ω_i over time interval $[t_{i-1}, t_i]$, for $1 \le i \le n$.

We assume that the incoming bitstream arrives at the input buffer at a constant rate of r bits/s and the playback device reads media objects from the playback buffer at a rate of C objs/s. Let the function $\alpha(k)$ denote the sum of bit length from the media objects 1 to k. Similarly, the function $F(k)$ denotes the sum of cycle numbers required to decode the media objects 1 to k. These two functions can be obtained by analyzing the given media bitstream.

The problem can be formally stated as follows. We assume that the first bit of the bitstream arrives at the input buffer at the instant of $t = 0$. Let $y^\pi(t)$ denote the number of processed media objects under the speed profile π during the time interval $[0, t]$. Given the input and playback buffer size b and B, respectively, cumulative cycle requirement $F(k)$, cumulative bit length $\alpha(k)$, and the bit length of a decoded media object U, what is the speed profile π achieving maximal energy savings while satisfying that: (1) the playback buffer never underflows and overflows; (2) the input buffer never underflows and overflows? This can be formulated as:

$$\text{Min} \sum_{i=1}^{\|\pi\|} P(\omega_i) \cdot (t_i - t_{i-1}),$$
$$\text{where } P(\cdot) \text{is the convex function on power–speed relationship} \tag{2.1}$$

s.t.

$$0 \le r \cdot t - \alpha(y^\pi(t)) \le b, \quad \forall t > 0 \tag{2.2}$$

$$\text{and } 0 \le \left(y^\pi(t) - C \cdot (t - t_d)\right) \cdot U \le B, \quad \forall t > 0,$$
$$\text{where } t_d : \text{ playback delay} \tag{2.3}$$

3 Energy optimization technique

Our solution to the problem (2.1) consists of three parts. The first part (Sect. 3.1) establishes the relationships between the constraints of buffers and the bounds of the processor speed. These relationships form the basis of the remaining two parts. The second part (Sect. 3.2) identifies the appropriate buffer sizes and playback delay parameters for a given video bitstream. These parameters will determine the existence of a feasible speed profile and the energy efficiency performance of the proposed scheme. Finally, given those parameters obtained by the second part, the third part (Sect. 3.3) computes the optimal speed profile.

3.1 Bounds on the processor speed

Given the cumulative bit length function $\alpha(k)$ and cumulative cycle requirement function $F(k)$, the buffer sizes b and B, we can compute the upper bound and lower bound of the processor speed as follows.

From (2.2) we have

$$r \cdot t - b \leq \alpha\left(y^{\pi}(t)\right) \leq r \cdot t \tag{3.1}$$

For the function $\alpha(k)$, we compute its inverse function $\alpha^{-1}(n)$, which returns an integer k. The sequence media frames of $[1, k]$ have the bit length n. Since $\alpha(k)$ refers to a cumulative process, both $\alpha(k)$ and $\alpha^{-1}(n)$ are monotonic increasing functions. By operating the inverse function $\alpha^{-1}(\cdot)$ on (3.1), we have:

$$\alpha^{-1}(r \cdot t - b) \leq y^{\pi}(t) \leq \alpha^{-1}(r \cdot t) \tag{3.2}$$

Moreover, we have the following relationship by performing the cumulative cycle requirement function $F(\cdot)$ on (3.2):

$$F\left(\alpha^{-1}(r \cdot t - b)\right) \leq F(y^{\pi}(t)) \leq F\left(\alpha^{-1}(r \cdot t)\right) \tag{3.3}$$

$F(y^{\pi}(t))$ is the exact cumulative frequency requirement function of the desired speed profile. Therefore (3.3) forms the upper and lower bounds on the processor speed under the constraint of the input buffer.

Similarly, we can derive the upper bound and lower bound of the processor speed under the constraint of the playback buffer according to (3.4):

$$F\left(C \cdot (t - t_d)\right) \leq F\left(y^{\pi}(t)\right) \leq F\left(C \cdot \left(t + T_p - t_d\right)\right), \quad \text{where } T_P = \frac{B}{C \cdot U} \tag{3.4}$$

We then combine (3.3) and (3.4) and form the global upper bound and lower bound of the processor speed:

$$\max\left(F\left(\alpha^{-1}(r \cdot t - b)\right), F\left(C \cdot (t - t_d)\right)\right) \leq F(y^{\pi}(t)) \leq \min\left(F\left(\alpha^{-1}(r \cdot t)\right), F\left(C \cdot (t + T_p - t_d)\right)\right) \tag{3.5}$$

3.2 Estimation of the input buffer and the playback buffer

The smoothing effect is closely related to the buffer size: larger buffer yields better smoothing performance. However, due to the shared computational resources and a significant source of energy consumption, it is desirable to reduce the allocated buffer to the application. Therefore, the estimation of the input buffer and the playback buffer becomes an important problem. We need to compute the appropriate buffer sizes in order to make the energy consumption resulted from the fluctuations below certain threshold.

Our bitstream analysis framework provides an insight into the buffering mechanism. For the sake of clarity, we denote the area enclosed by (3.3) as S_{in}, the area by (3.4) as S_{pl}, and the area by (3.5) as S_{gb}. In other words, $S_{in}(S_{pl})$ represents the range of processor speeds that will not lead to underflow or overflow of the input buffer (playback buffer). S_{gb} is the range of processor speeds which is allowed by both the input buffer and playback buffer. As shown in (3.3), S_{in} has one tunable parameter: input buffer size b, where larger input buffer can increase the area of S_{in}. From (3.4), S_{pl} is controlled by two parameters: the playback buffer size B which controls its area, and the playback delay t_d that controls its position. According to (3.5), S_{gb} can be geometrically interpreted as the intersection of S_{in} and S_{pl}. Larger S_{gb} is desired, since more space is allowed for the smoothing operation. According to the above analysis, we can shape S_{gb} by tuning the three parameters b, B and t_d. The effects of b and B are straightforward. On the other hand, the relative position between S_{in} and S_{pl} plays an important role in shaping S_{gb}: when the upper bound of playback buffer aligns with the upper bound of input buffer, S_{gb} yields the maximal space for the smoothing algorithm with minimal playback delay.

Our estimation algorithm consists of two parts. The first part computes the buffer sizes based on the feasibility conditions, which refer to the minimal buffer requirements to guarantee the existence of a feasible speed profile. This is the basic requirement for a media decoding process. When the buffer sizes of feasibility conditions cannot provide a satisfactory smoothing effect, we need to invoke the second part. The principle of the second part is straightforward. It increases the buffer sizes in fine steps, and then computes the speed profile using the optimal speed profile algorithm (details can be found in Sect. 3.3) for the given buffers. From the speed profile, we can compute the energy consumption level resulted from the fluctuations. This process repeats until the energy consumption resulting from the fluctuations is below some threshold. Though the second part will incur intensive computations as it employs an exploration strategy, the experimental results in Sect. 4.2 show that this part is usually unnecessary since the buffer sizes of feasibility conditions are sufficient for the applications. We believe that this stems from the large fluctuation levels of sizes and workload among the video bitstream: the buffer sizes, which just satisfy the feasibility conditions of the worst case, can work satisfactorily for the entire bitstream.

3.2.1 Feasibility conditions

To guarantee that the global bounds in (3.5) have feasible speed profiles, B and b should satisfy certain conditions. We call these feasibility conditions. They are subject to the following two constraints: the lower bound of the input buffer should be less than the upper bound of the playback buffer (3.6), and the lower bound of playback buffer should be less than the upper bound of input buffer (3.7).

$$F\left(\alpha^{-1}(r \cdot t - b)\right) \leq F\left(y^{\pi}(t)\right) \leq F\left(C \cdot (t + T_p - t_d)\right) \tag{3.6}$$

$$F\left(C \cdot (t - t_d)\right) \leq F\left(y^{\pi}(t)\right) \leq F\left(\alpha^{-1}(r \cdot t)\right) \tag{3.7}$$

From (3.6) and (3.7), we can derive feasibility conditions for B and b. Let $\Delta = T_p - t_d$, which indicates the position of the upper bound of playback buffer, and Δ_0 denote the desirable value that the upper bound of playback buffer aligns with the upper bound of input buffer.

One possible way to calculate Δ_0 is as (3.8):

$$\Delta_0 = \arg\min_{\Delta} \sum_{t \geq 0} \left| F\left(C \cdot (t + \Delta)\right) - F\left(\alpha^{-1}(r \cdot t)\right) \right| \tag{3.8}$$

From (3.6) and (3.7) we have:

$$\begin{cases} b \geq r \cdot t - \alpha\left(C \cdot (t + \Delta_0)\right), & t \geq 0 \\ B \geq C \cdot U \cdot (t + \Delta_0) - \alpha^{-1}(r \cdot t) \cdot U, & t \geq 0 \end{cases} \tag{3.9}$$

To fully utilize the buffers, we can use the larger one of these two feasibility sizes for both the input and playback buffers. This equals to shifting the lower bound of the smaller buffer towards right side to align with the lower bound of the larger buffer. Towards this, we need to convert b and B into the numbers of media objects $\bar{b}$ and $\bar{B}$ respectively, as shown in (3.10). It is noted that we cannot directly manipulate the bit length results from (3.9), since they are associated with the compressed data and the decompressed data, respectively. After aligning the lower bounds of input buffer and playback buffer, we can compute their modified bit length b' and B' in (3.10).

$$\begin{cases} \bar{b} = \alpha^{-1}(b), & b' = \alpha(\max(\bar{b}, \bar{B})) \\ \bar{B} = B/U, & B' = \max(\bar{b}, \bar{B}) \cdot U \end{cases} \tag{3.10}$$

Finally we can calculate the modified playback delay as:

$$t'_d = T'_p - \Delta_0, \quad \text{where } \Delta_0 \text{ is given by (3.8) and}$$
$$T'_p = \frac{B'}{C \cdot U} \tag{3.11}$$

3.3 The optimal speed profile algorithm

Given the global bounds of the processor speed obtained from Sect. 3.2, we have developed the optimal speed profile (OSP) algorithm.

To reduce the fluctuation level of the generated speed profile, the key idea behind the proposed algorithm is to make the speed profile close to the mean value of the processing workload. This principle has two implications. First, when the average workload is feasible for a given segment, we use it as the speed profile of the segment since its fluctuation level is minimal. Second, when the processor speed must be changed to ensure feasibility, we change the processor speed based on the largest deviation points from the average workload, since it is the closest feasible speed profile to the average speed.

We illustrate the critical part of the proposed algorithm in Fig. 2: given the global bounds, our algorithm is required to find a smoothing speed profile over period $[O, A]$. We first construct a straight line OA since it achieves maximal energy saving. However, this is infeasible as OA violates the global bounds near points B and C. Among those violated bounds, we identify point B having the largest deviation from line OA. Then the speed profile is split into two parts, line OB and line BA, to satisfy the speed bounds at B. The new speed profile may violate global bounds as well, such as point C in Fig. 2. We then perform the same splitting process resulting in line OC and CB. The process continues iteratively until all violations are eliminated.

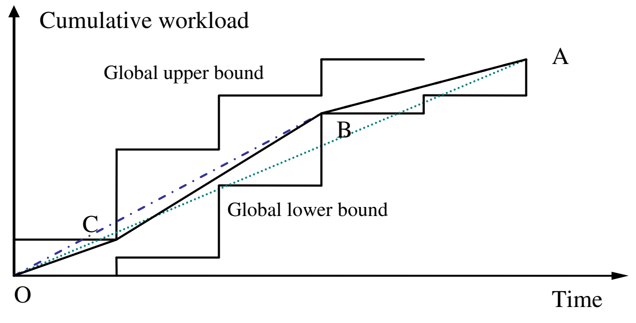

Fig. 2 Illustration of the optimal speed profile algorithm

We give the detailed description of the proposed algorithm in Fig. 3.

Concerning the performance of the optimal speed profile algorithm, we have the following theorem.

Theorem 3.1 *Given the same configurations of the algorithm, the speed profile generated by the optimal speed profile algorithm achieves the minimal energy consumption among all feasible speed profiles.*

Proof See Sect. 6. □

4 Experimental results

In this section, we present two kinds of experimental results concerning the proposed scheme. The first is the comparison with the representative buffer-based DVS schemes, which is used to evaluate the effectiveness of the proposed scheme. The second includes: (1) the comparison with the theoretical minimal energy consumption of the given bitstream; and (2) the overhead of the speed profile, both of which provide insights into the properties of our scheme.

We carried out experiments on six video clips selected from the MPEG test dataset: Akiyo, Highway, Coastguard, Container, Hall and Walk. All of them have an identical resolution of 352 × 288 and frame rate of 30 fps. They are encoded in MPEG-2 format. We employed The Core Pocket Media Player (TCPMP) [17] as the decoder application, since it is an open-source media player optimized for mobile devices.

```
The Optimal Speed Profile Algorithm
INPUT: $B_{\max}(\cdot)$: global upper bound of the speed
       $B_{\min}(\cdot)$: global lower bound of the speed
OUTPUT: $\Omega$: set of speed profile triple: $\langle t_s, t_e, \omega \rangle$ processor
        speed $\omega$ over period $[t_s, t_e]$
FUNCTIONS:
    RecursiveSmoothing$(t_s, w_s, t_e, w_e)$ : to find a smoothing speed
        profile from time $t_s$ with accomplished workload $w_s$
        to time $t_e$ with workload $w_e$
    $\max_w(s)$ : the maximal workload in sequence $s$
    $\max_t(s)$ : time index of the maximal workload in $s$
BEGIN
  $\Omega$ = RecursiveSmoothing$(t_{in}, 0, \max_t(B_{\min}), \max_w(B_{\min}))$
  $\Omega \leftarrow \Omega + \langle 0, t_{in}, 0 \rangle$
END

FUNCTION RecursiveSmoothing$(t_s, w_s, t_e, w_e)$
$\Gamma_{\max} \leftarrow \varnothing$, $\Gamma_{\min} \leftarrow \varnothing$, $t \leftarrow t_s$
WHILE $t \le t_e$
$d_{\max} \leftarrow B_{\max}(t) - ((t - t_s)(w_e - w_s)/(t_e - t_s) + w_s)$
$d_{\min} \leftarrow B_{\min}(t) - ((t - t_s)(w_e - w_s)/(t_e - t_s) + w_s)$
    IF $d_{\max} < 0$
        $\Gamma_{\max} \leftarrow \Gamma_{\max} \cup (t, |d_{\max}|)$
    ELSEIF $d_{\min} > 0$
        $\Gamma_{\min} \leftarrow \Gamma_{\min} \cup (t, d_{\min})$
    ENDIF
$t \leftarrow t + 1$
ENDWHILE
  IF $\Gamma_{\max} = \varnothing$ AND $\Gamma_{\min} = \varnothing$
      $\Omega \leftarrow \langle t_s, t_e, (w_e - w_s)/(t_e - t_s) \rangle$
ELSE
    IF $\max_w(\Gamma_{\max}) > \max_w(\Gamma_{\min})$
        $t_m = \max_t(\Gamma_{\max})$, $w_m = B_{\max}(t_m)$
    ELSE
        $t_m = \max_t(\Gamma_{\min})$, $w_m = B_{\min}(t_m)$
    ENDIF
  $\Omega_1$ = RecursiveSmoothing$(t_s, w_s, t_m, w_m)$
$\Omega_2$ = RecursiveSmoothing$(t_m, w_m, t_e, w_e)$
$\Omega \leftarrow \Omega_1 + \Omega_2$
ENDIF
RETURN $\Omega$
```

Fig. 3 The optimal speed profile algorithm

We measured power consumption of the video decoding application using a development board of "RainboW" [18], which is designed to meet a wide spectrum of mobile applications. It consists of the Intel PXA 270 that is based on an enhanced version of the Intel XScale, 64 MB SDRAM, 64 MB Flash, 3.5 in. QVGA with Touch Panel, QWERTY keyboard, USB ports, wireless connectivity, and audio video components, etc. This system runs Linux 2.6 ported for ARM. The development board has been customized with jumper wires to allow current and voltage measurements for main components of the system. We used National Instruments PXI-4071 $7\frac{1}{2}$-digit digital multimeters to measure our target components. These multimeters were then connected to a desktop computer to record the collected data. We focused on power supply of the processor since it is targeted by DVS techniques.

To obtain an accurate estimation on required buffer sizes for the bitstream analysis, it was necessary to perform the algorithm using a fine granularity. Because of this, we chose Macroblock as the basic media object in line with [19]. We obtained the computational workload of each Macroblock using a simulation method which is based on ARM architecture [20].

We implemented our scheme with the buffer sizes of feasibility condition. Towards this, we computed the speed profile as follows. Given the workload and bit length trace of a video clip, we first computed the feasible sizes of the input and playback buffers according to (3.9). We then performed (3.10) to fully utilize the buffers and computed the playback delay according to (3.11). Finally we computed the speed profile using OSP algorithm with those obtained parameters. The resultant configurations for the six video clips are listed in Table 1. We made use of these configurations to conduct the experiments in Sects. 4.1 and 4.2.

4.1 Comparisons with existing buffer-based speed control schemes

Since energy efficiency and buffer requirement well characterize a buffer-based DVS scheme when it provides guaranteed QoS for video decoding, we compared our scheme with two representative existing buffer-based DVS schemes to investigate: (1) energy consumption at the same buffer level, and (2) the maximal buffer occupancy at the same energy consumption level. We used dithering to convert the required continuous voltage levels to discrete levels provided by PXA 270, which is widely employed in such case [21]. The baseline 1 is based on [14], which employs buffers to reduce the idle processor periods resulting from the large variation of video decoding time. For fairness, we set the output buffer size of baseline 1 as the sum of the input and playback buffer sizes of our scheme and execute both algorithms at the Macroblock level. We showed the energy consumption ratios between our scheme and the baseline 1 in Fig. 4. Compared to the baseline 1, our scheme can achieve 22.8% energy savings on average with the same buffer sizes.

The baseline 2 is based on feedback control with PI controller [11], which scales the processor speed by monitoring the filling level of the playback buffer. To keep its energy con-

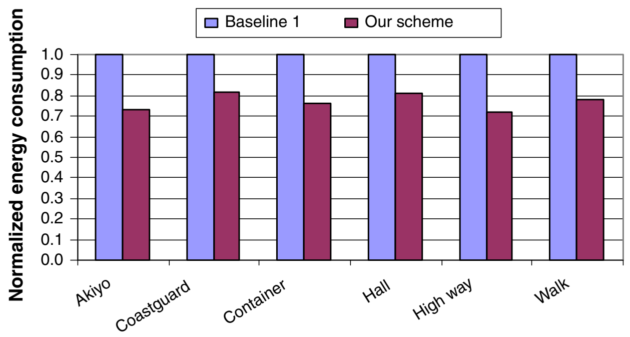

Fig. 4 Normalized energy consumption between our scheme and the baseline 1 for the six test video clips

sumption close to the optimal energy consumption at Macroblock level, we scaled the controller parameters from frame level by the number of Macroblocks in a frame, namely, the proportional factor and integral factor (0.01 and 0.0145, respectively, according to the author's suggestion) were scaled down by 396 (namely, the number of Macroblocks contained in a frame), dead-zone, forward window and feedback window were scaled up by 396. The energy consumption ratio and the required buffer sizes are listed in Table 2. From Table 2, we can see that at the similar energy consumption level, our method can achieve 51.2% reduction in buffer requirement on average.

Baselines 1 and 2 represent the recent advances in buffer based DVS techniques. The above experimental results show that the proposed scheme significantly outperforms existing DVS schemes. The improvements achieved by the proposed scheme can be interpreted in two alternative ways. First, due to the elimination of QoS interference, the energy efficiency of smoothing mechanism has been increased by 22.8%. Secondly, with accurate workload estimation, only a half size of the buffer is needed to provide the guaranteed QoS. These demonstrate the superiority of the proposed media server supported DVS scheme and show its high potentials for low-power video decoding applications.

4.2 A Comparison with the theoretical minimal energy consumption and the overhead of speed profiles

Due to the limited sizes of buffers, the proposed scheme leads to more energy consumption than that of the theoretical minimal energy consumption (TMEC), which is the energy consumption corresponding to the global average speed, and is computed as $T \cdot P(\bar{\omega})$, where $\bar{\omega} = \frac{1}{T}\sum_{t\geq 0}\omega(t)$, $\{\omega(t)|0 \leq t \leq T\}$ is the required speed for each media object. Intuitively, we can increase the buffer sizes to further reduce the energy consumption. It is then important to investigate the following questions: (1) how much further energy savings can be achieved with the increased buffer sizes? (2) what is the relationship between the increase in buffer size and the improvement of energy efficiency? (3) what are the appropriate buffer sizes for a given media clip?

To answer these questions, for each test video clip we first compared the energy consumption of TMEC with our scheme using the minimal requirement of the input and playback buffers. The results are summarized in Table 3. Then we increased the buffer sizes and computed their corresponding energy consumptions, as illustrated in Fig. 5, to show the relationship between the increase in buffer size and the improvement of energy efficiency.

These results have two important implications. Using the configurations in Table 1, namely, 14.40 K bytes of the input buffer, 453.48 K bytes of the playback buffer on average, the energy consumption of the proposed scheme is very close to the theoretical lower bound with only 1.2–2.2% additional overhead. The results suggest that the proposed scheme works well with sufficiently small-sized buffers. This also implies that increasing the buffer sizes or prolonging the latency will hardly reduce the energy consumption further. This is supported by the results shown in Fig. 5, where we only achieved less than 2% additional energy reduction while increasing the buffer sizes by 33–50%. These facts show that the buffer sizes of feasibility conditions are appropriate for the video decoding applications.

Table 2 Comparisons between our scheme and the baseline 2

Clip name	Akiyo	Coastguard	Container	Hall	Highway	Walk
NEC	1.04	1.06	1.01	1.02	1.03	1.03
BUF	1,999	2,436	2,021	2,047	2,171	3,565
RED	0.474	0.578	0.488	0.497	0.425	0.607

NEC Normalized energy consumption of the baseline 2 over our scheme, *BUF* maximal buffer occupancy of the baseline 2 in terms of Macroblocks, *RED* reduced buffer size ratio achieved by our scheme (refer to Table 1)

Table 3 Energy consumption ratio between the proposed scheme and the TMEC

Clip name	Akiyo	Coastguard	Container	Hall	Highway	Walk
TMEC	1.012	1.022	1.016	1.016	1.012	1.018

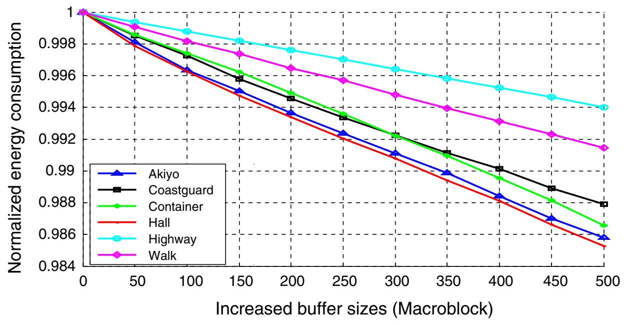

Fig. 5 Normalized energy consumption with the buffer sizes increased from the feasibility condition for the six video clips

Another important property of our scheme is the size of the corresponding speed profile, which is determined by the required number of scaling operations. In our scheme, the processor speed is kept as a constant until the speed has to be changed due to the constraints of the buffers. This allows us to add a new item to the side information only when the processor speed is changed. The numbers of scaling operations of the six video clips at Macroblock level are listed in Table 4.

As shown in Table 4, the numbers of scaling operations range from 14 to 55. For each scaling operation, the speed profile is added as an item with two parts: the Macroblock index where the processor speed needs to be scaled, and the new speed level. We estimate the speed profile size as follows. Assuming that 28 bits are used to represent the index of the Macroblocks, which is sufficient for 110 min video clips with frame rate of 30 fps and resolution of 720×480 (DVD standard), and 6 bits are used to identify the speed levels (namely, 64 speed levels), the corresponding speed profile sizes are listed in the third row of Table 4. Compared to the content sizes (the fourth row of Table 4), the overhead associated with side information is completely negligible.

5 Conclusion and future work

We have proposed a new concept of media server supported DVS, which is superior to existing client-only approaches. The key ideas include supporting energy efficiency by smoothing mechanism and guaranteeing playback QoS by bitstream analysis. As a first step to this new direction, we have proposed an optimal speed control scheme for intra-task voltage scheduling, in line with [3], which adopts single-task mode and is suitable for dedicated mobile devices. The significance of our scheme is twofold. First, it lays a solid theoretical foundation for this direction of research. It can serve as a generic model which is possible to be extended to solve various problems. Second, it improves the performance of DVS in terms of energy efficiency or memory requirements significantly. More importantly, our scheme facilitates practical applications from four aspects: (1) it does not require feedback information from the clients; (2) a single speed profile can yield satisfactory performance of energy efficiency; (3) the required buffer sizes can be met by most mobile devices; and (4) the sizes of resultant speed profiles are negligible compared to the media content. These properties render our scheme suitable for a wide range of applications from download-playback and video on demand, to broadcast and multicast.

For the future work, the proposed scheme can be extended to multiple task scenarios. Moreover, our analysis framework provides more accurate estimation of input and playback buffer configurations in terms of individual media bitstreams. The new estimation can be used to identify the build-in buffer ranges to support *a class* of multimedia processing applications, which is an important issue in designing a SoC platform [19].

6 Proof of optimality

Lemma 1 *For a convex function $P()$, if $b > a$, then we have $P(b+c)-P(b)>P(a+c)-P(a)$, for $c > 0$*

Consider a period $[T_s, T_e]$ in the decoding of a media bitstream, we define the instant T_s and T_e as the beginning point and the end point of the period, and the sum of workload between T_s and T_e as the cumulative workload.

Lemma 2 *For a given media bitstream, two speed profiles, with the same cumulative workload between the beginning point and the end point of a period, will have the same average speed during this period.*

Lemma 3 *For a certain cumulative workload of a period $[T_s, T_e]$, the average speed over $[T_s, T_e]$ incurs the minimal energy consumption.*

Table 4 The number of scaling operations, corresponding speed profile sizes, and content sizes for the six video clips

Clip name	Akiyo	Coastguard	Container	Hall	Highway	Walk
Scaling number	21	14	22	35	55	34
Speed profile (byte)	90	60	94	149	234	145
Content (Mbyte)	1.33	1.33	1.33	1.33	9.03	1.59

For Lemma 1, according to the definition of convex function, we have:

$$\begin{aligned}P(b) &\leq \frac{P(b+c)-P(a)}{b+c-a}\cdot(b-a)+P(a)\\ &= \frac{(b-a)\cdot P(b+c)+cP(a)}{b+c-a}\\ P(a+c) &\leq \frac{P(b+c)-P(a)}{b+c-a}\cdot c+P(a)\\ &= \frac{cP(b+c)+(b-a)\cdot P(a)}{b+c-a}\end{aligned}$$

By adding the above two inequations, we have:

$$P(b)+P(a+c)\leq P(b+c)+P(a)$$

Then we have

$$P(b+c)-P(b)\geq P(a+c)-P(a).$$

Since Lemma 2 and 3 are quite straightforward, we skip their proofs.

We define change points as those instants when the speed of the processor is changed. For example, points A, B and C are all change points in Fig. 2. Considering $S^* = \{\omega^*(j)|1 \leq j \leq N\}$, which is the speed profile generated by the optimal speed profile (OSP) algorithm, all the change points of S* can be divided into two classes: (1) min-change points: the changes of processor speeds are due to global lower bound; and (2) max-change points: the changes of processor speeds are due to global upper bound.

Then, we can derive that for any feasible speed scheme at a change point of S*, it must be not lower than min-change points and not higher than max-change points. We can see that the beginning point of the speed scheme is a max-change point and the end point of the speed scheme is a min-change point. Considering a feasible speed profile, its cumulative workload will only intersect that of S* at the following points: beginning point, end point, some points between neighboring pair of min-change points and max-change points. We call these intersection points. The intersection points divide the whole speed scheme into various segments. Each segment only contains homogeneous change points.

Next we will show that in each segment, our scheme is the optimal one among all feasible speed schemes. Then, we can conclude that our scheme is optimal for the full range of speed profiles. We first consider a segment which contains the min-change points. We construct a speed profile S=$\{\omega(n)|1 \leq n \leq N\}$, subject to: (1) its cumulative workload intersects that of S* at the beginning and end points of the segment; (2) it only changes speeds at the change points of S*; (3) within the segment, the cumulative workload of S is not less than that of S*. It should be noted that S is not necessarily a feasible speed profile.

In a segment which contains min-change points, S* has an important property: its speeds are mono-decreasing. We show this property as follows. According to OSP algorithm, the S* is generated based on the deviation from the average value of the speed of some region. As illustrated in Fig. 6a, let point A have the largest deviation from the average value of the segment, point B have largest deviation from the average value of region $[O, A]$, and points C, D have the largest deviation from the average value of region $[O, B]$ and $[B, A]$, respectively. In our algorithm, we first split the region $[O, A]$ into $[O, B]$ and $[B, A]$, then split $[O, B]$ and $[B, A]$. After that, we have obtained the resultant speed profile, as shown in solid line in Fig. 6a. We denote the cumulative workload of B, C, and D as x, y, z, temporal offsets from point O for B, C, and D as c, b, d, and the average speed over $[O, A]$ as r. Since B has the largest deviation from the average speed, we have $x-r\cdot b > y-r\cdot c$, and $x-r\cdot b > z-r\cdot d$. Then we have $\frac{x-y}{b-c} > r > \frac{z-x}{d-b}$. It is noted that $(x-y)/(b-c)$ is the speed over $[C, B]$ and $(z-x)/(d-b)$ is the speed over $[B, D]$. This relationship holds for each splitting operation performed in the algorithm. This proves the property: for a segment which contains min-change points, the speeds of S* are mono-decreasing. Similarly, we can prove that for a segment which contains max-change points, the speeds of S* are mono-increasing.

Based on the construction rules, the cumulative workload of S is not less than that of S*, as shown in Fig. 6b:

$$\text{We have } \sum_{j=1}^{n}\omega(j)\cdot\tau(j)\geq\sum_{j=1}^{n}\omega^*(j)\cdot\tau(j),\quad 1\leq n\leq N \tag{6.1}$$

For region $\tau(j)$, we denote $\Delta\omega(j)=\omega(j)-\omega^*(j)$.

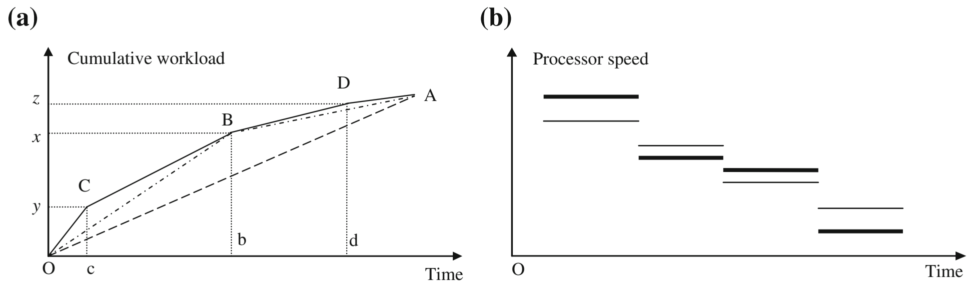

Fig. 6 **a** Illustration of the splitting operation; **b** Illustration of the speed profiles, *thin lines* stand for speed profile S*, *thick lines* stand for speed profile S

From (6.1), we have

$$\sum_{j=1}^{n} \Delta\omega(j) \cdot \tau(j) \geq 0, \quad 1 \leq n \leq N \tag{6.2}$$

We divide
$$\begin{cases} \Delta\omega^{+}(j) = \omega(j) - \omega^{*}(j), & \omega(j) > \omega^{*}(j) \\ \Delta\omega^{-}(j) = \omega^{*}(j) - \omega(j), & \omega(j) < \omega^{*}(j) \end{cases} \tag{6.3}$$

We denote
$$\begin{cases} G^{+}(i) = \{k \mid k \leq i, \quad \omega(k) > \omega^{*}(k)\} \\ G^{-}(i) = \{k \mid k \leq i, \quad \omega(k) < \omega^{*}(k)\} \end{cases} \tag{6.4}$$

There must exist an partition R,

$$R = \left\{ r_j(i) \left| \begin{array}{lll} r_j(i) \subset \tau(i), & r_j(i) \cap r_k(i) = \emptyset, & i \in G^{+}(j), \\ j \neq k, & \text{and} j, \quad k \in G^{-}(N), & \text{and} j, \quad k \notin G^{-}(i) \end{array} \right. \right\}$$

For any $j \in G^{-}(j)$, we have:

$$\Delta\omega^{-}(j) \cdot \tau(j) \leq \sum_{i=1}^{j-1} \Delta\omega^{+}(i) \cdot r_j(i) \tag{6.5}$$

Next we prove (6.5) by contradiction. We suppose that some $\Delta\omega^{-}(k)$ cannot be covered. Then for k, we have $\sum_{j=1}^{k} \Delta\omega(j) \cdot \tau(j) < 0$, this contradicts with (6.2).

As the power is a convex function of the speed of the processor, substitute Lemma 1 into (6.5), we have:

$$\sum_{i \in G^{+}(i)} P(\omega(i)) - P\left(\omega^{*}(i)\right) \cdot \tau(i)$$
$$> \sum_{j \in G^{-}(j)} P\left(\omega^{*}(j)\right) - P(\omega(j)) \cdot \tau(j) \tag{6.6}$$

This shows that the constructed speed profile S will have larger energy consumption than the speed profile S* for the given segment.

Now we consider the segment which only contains max-change points. We construct a speed profile S, which is subject to: (1) its cumulative workload intersects that of S* at the beginning and end points of the segment; (2) it only changes speeds at the change points of S*; (3) within the segment, the cumulative workload of S is not greater than that of S*. Thus we have:

$$\sum_{j=1}^{n} \omega(j) \cdot \tau(j) \leq \sum_{j=1}^{n} \omega^{*}(j) \cdot \tau(j), \quad 1 \leq n \leq N \tag{6.7}$$

On the other hand, from Lemma 2, we have:

$$\sum_{j=1}^{n} \omega(j) \cdot \tau(j) + \sum_{j=n+1}^{N} \omega(j) \cdot \tau(j)$$
$$= \sum_{j=1}^{n} \omega^{*}(j) \cdot \tau(j) + \sum_{j=n+1}^{N} \omega^{*}(j) \cdot \tau(j) \tag{6.8}$$

We consider the situation from the end of the segment.

$$\sum_{j=N}^{n} \omega(j) \cdot \tau(j) \geq \sum_{j=N}^{n} \omega^{*}(j) \cdot \tau(j), \quad 1 \leq n \leq N \tag{6.9}$$

As shown above, the speeds are mono-increasing for such segment. Thus we have the same result for segment which only contains the max-change points.

Finally, we consider a feasible speed profile S′. A feasible speed profile must be not less than a min-change point and not greater than a mix-change point and intersects the cumulative workload of S* at intersection points. But S′ can use arbitrary feasible curves to connect these points. Corresponding to S′, we can construct a speed profile S with the same cumulative workload at the change points and the intersection points, using straight lines to connect those points. According to Lemma 3, the energy consumption of S′ is not less than that of S. And we have shown that the energy consumption of S* is not greater than that of S. Therefore, we can conclude that the energy consumption of S* is not greater than that of any feasible speed profile.

Acknowledgments We thank Dr. Samarjit Chakraborty for his help in power consumption measurements, and Mr. Vu An Tran for assistance in running some experiments. We also wish to thank the anonymous reviewers for their critical comments, which have helped to improve the quality of this paper. The work carried out was funded by the Singaporean Ministry of Education under the research grant R-252-000-236-112.

References

1. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. IEEE Annual Foundations of Computer Science. pp. 374–382 (1995)
2. Ishihara, T., Yasuura, H.: Voltage scheduling problem for dynamically variable voltage processors. In: International Symposium on Low Power Electronics and Design, pp. 197–202 (1999)
3. Shin, D., Kim, J., Lee, S.: Intra-task voltage scheduling for low-energy hard real-time applications. IEEE Des. Test. Comput. **18**(2), 20–30 (2001). doi:10.1109/54.914596
4. Weiser, M., Welch, B., Demers, A., Shenker, S.: Scheduling for reduced CPU energy. Oper. Syst. Des. Implement. 13–23 (1994)
5. Pering, T., Burd, T., Broderson, R.: The simulation and evaluation of dynamic voltage scaling algorithms. In: International Symposium on Low Power Electronics and Design, pp. 76–81 (1998)
6. Choi, K., Dantu, K., Cheng, W., Pedram, M.: Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In: International Conference on Computer Aided Design, pp. 732–737 (2002)
7. Bavier, A., Montz, A., Peterson, L.: Predicting MPEG execution times. In: SIGMETRICS/PERFORMANCE International Conference on Measurement and Modeling of Computer Systems, pp. 131–140 (1998)
8. Yuan, W., Nahrstedt, K.: Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In: ACM Symposium on Operating Systems Principles, pp. 149–163 (2003)
9. Chung, E., Benini, L., Micheli, G.: Contents provider-assisted dynamic voltage scaling for low energy multimedia applications. ISLPED, pp. 42–47 (2002)

10. Nielsen, L.S., Niessen, C., Sparso, J., Van Berkel, K.: Low-power operation using self timed circuits and adaptive scaling of the supply voltage. IEEE Trans. VLSI Syst. **2**(4), 391–397 (1994). doi:10.1109/92.335008
11. Zhijian, L., John, L., Mircea, S., Kevin, S.: Design and implementation of an energy efficient multimedia playback system. ACSSC, pp. 1491–1497 (2006)
12. Gutnik, V., Chandrakasan, A.P.: Embedded power supply for low power DSP. IEEE Trans. VLSI Syst. **5**(4), 425–435 (1997). doi:10.1109/92.645069
13. Lu, Y.H., Benini, L., Micheli, G.D.: Dynamic frequency scaling with buffer insertion for mixed workloads. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **21**(11), 1284–1305 (2002). doi:10.1109/TCAD.2002.804087
14. Im, C., Ha, S., Kim, H.: Dynamic voltage scheduling with buffers for low-power multimedia applications. ACM Trans. Embed. Comput. Syst. **3**(4), 686–705 (2004). doi:10.1145/1027794.1027796
15. Lorch, J.R., Smith, A.J.: PACE: A new approach to dynamic voltage scaling. IEEE Trans. Comput. **53**(7), 856–869 (2004). doi:10.1109/TC.2004.35
16. Xiaobo, F., Carla, S., Alvin, R.: Memory controller policies for DRAM power management. In: International Symposium on Low Power Electronics and Design, pp. 129–134, August 2001
17. http://tcpmp.corecodec.org/
18. http://www.iwavesystems.com
19. Liu, Y., Maxiaguine, A., Chakraborty, S., Ooi, W.T.: Processor frequency selection for SoC platforms for multimedia applications. In: IEEE Real-Time Systems Symposium, Lisbon, pp. 336–345, December 2004
20. SimpleScalar/ARM: http://www.simplescalar.com/v4test.html
21. Luo, J., Jha, N.K.: Power-profile driven variable voltage scaling for heterogeneous distributed real-time embedded systems. In: International Conference on VLSI Design, pp.369–375 (2003)