

Dynamic Voltage Scaling on MPEG Decoding¹

Donghwan Son,
Electronics and Telecommunications
Research Institute
161 Kajong, Yusung, Taejon, 305-
350 Korea
dhson @ etri.re.kr

Chansu Yu,
Information and
Communications University
58-4 Hwaam, Yusung,
Taejon, 305-732, Korea
cyu @ icu.ac.kr

and Heung-Nam Kim
Electronics and Telecommunications
Research Institute
161 Kajong, Yusung, Taejon, 305-
350 Korea
hnmkim @ etri.re.kr

Abstract

A number of research efforts have been devoted to reduce energy consumption of a processor without impacting the performance through the use of *dynamic voltage scaling (DVS)*. This paper presents two DVS algorithms on MPEG decoding. One is *DVS with delay and drop rate minimizing algorithm (DVS-DM)* where voltage is determined based on previous workload only. Another algorithm scales the supply voltage according to the predicted MPEG decoding time and previous workload (*DVS with predicted decoding time or DVS-PD*). Simulation results show that DVS-PD improves energy efficiency as much as 56% compared to the conventional *shutdown algorithm*. We also found that the amount of energy saving with DVS-PD is not affected by the fluctuation of the movie stream. But it is related with error rate of the predictor, which implies that if decoding time is predicted more accurately, DVS algorithm can be more efficient.

1. Introduction

As mobile devices require more computation as well as communication activities, energy efficiency of a processor becomes the most critical because they may have neither display devices nor disks. The most effective way to reduce power consumption of a processor core in CMOS technology is to lower the supply voltage, which exploits the quadratic dependence of power on voltage. Reducing the supply voltage however increases circuit delay and decreases clock speed and thus, it may not be effective because some systems have latency critical tasks. One possible solution is to dynamically vary the voltage according to the processor workload. Recent advances in

power supply technology make it possible to create processor cores with varying supply voltages according to application's time constraints [1,2]. Current custom and commercial CMOS chips are capable of operating reliably over a range of supply voltages and efficient variable voltage DC suppliers are available [3,4].

Dynamic voltage scaling (DVS) allows a processor to dynamically change its speed and voltage at run time, increasing energy efficiency. In [5-7], *interval based DVS* has been proposed, which divides time into uniform-length intervals and analyzes system utilization of the previous intervals to determine the voltage of the next interval accordingly. Interval based scheduling is simple and easy to implement, but it often incorrectly predicts future workloads and degrades the quality of service. For real-time systems, there proposed several DVS algorithms that minimize energy consumption while all tasks are guaranteed to complete on or before deadlines [8-11].

On the other hand, the mobile computing environment has been changed rapidly in recent years. One of the variations is the growing number of multimedia applications, ranging from video games and movie players to sophisticated virtual reality environment on mobile devices. MPEG decoder may be one of those applications and will be widely adopted in mobile devices from notebook computers to communication devices. The core of MPEG decoder is an *inverse discrete cosine transform (IDCT)* which decompresses MPEG video frames. From frame to frame, required decompression work varies widely due in part to the fact that a given MPEG video stream contains different frame types, and in part to the potential wide variation between scenes. However, each frame must be processed in time according to the frame rate of the stream. The variability of the workload and the realtime constraint causes the interval based DVS algorithm to fail since it is difficult to predict the next workload based on the previous workload and a wrong prediction causes frames to be dropped.

¹ This research was supported in part by the Electronics and Telecommunications Research Institute under Grant No. 00104

In this paper, we propose two DVS algorithms on MPEG decoding. The first algorithm is *DVS-DM* (*DVS with delay and drop rate minimizing algorithm*), which is a kind of interval based DVS in a sense that it schedules voltage based on previous workload. Another algorithm is *DVS-PD* (*DVS with decoding time prediction*), which determines the voltage not only by previous workload but also by predicted MPEG decoding time. The prediction, in this case, is based on frame size and frame type as suggested in [12], and the resulting better prediction leads to noticeable energy saving as much as 56% compared to the conventional shutdown algorithm explained in Section 2.1. The rest of the paper is organized as follows. Section 2 presents the related works on DVS and MPEG decoding. Two DVS algorithms are proposed in Section 3. Experimental environments as well as simulation results are presented in Section 4. Conclusion remarks are found in Section 5.

2. Related Works

2.1. Dynamic Voltage Scaling (DVS)

In a digital static CMOS circuit, energy consumed by each operation is given by the following equation [6]

$$E \propto C_{eff} V^2 f_{CLK}$$

where C_{eff} is the effective switching capacitance of the operation, V is the supply voltage, and f_{CLK} is the clock frequency. On the other hand, delay of the circuit and the clock frequency are given by:

$$delay \propto \frac{V}{(V - V_k)^\alpha} \quad \text{and} \quad f_{CLK} \propto \frac{(V - V_k)^\alpha}{V}$$

where α ranges from 1 to 2, and V_k depends on threshold voltage at which velocity saturation occurs [6]. Regardless the exact value of V_k , the combination of the two equations implies that lowering the supply voltage will result in longer delay, but at the same time quadratic decrease in energy. In reality, it is more than quadratic since the clock frequency can also be reduced when the supply voltage is scaled down. Current VLSI technology supports a set of supply voltage levels. For example, a *Motorola CMOS 6805* microcontroller is rated at 6Mhz at 5.0 volts, 4.5Mhz at 3.3 volts, and 3Mhz at 2.2 volts [5], and *embedded SL enhanced Intel486 DX2* processors run 66Mhz at 5 volts and 50Mhz at 3.3 volts. More recently, *Crusoe* processor from *Transmeta* has 16 voltage levels and *LongRun* power management is provided to scale voltage on application side [13]. This contrasts to other power saving schemes where the operating frequency is changed instead of the supply voltage. If only the frequency were changed, instantaneous energy consumption is altered, but the total energy consumed for each operation remains the same.

Dynamic Voltage Scaling (DVS) is a technique that

allows a voltage scheduler to alter a microprocessor's operating voltage at run-time and thus trade-off energy for delay [5,6]. Since the workload is usually not constant but in bursts, there are idle times in which CPU does not work [14]. One conventional way of reducing CPU power consumption is *shutdown mechanism*, which stops CPU operation while idling. Comparing with DVS, it is less effective since the energy is a quadratic function of voltage. Figure 1(a) shows an example of the voltage-scheduling graph with the shutdown mechanism. The X-axis represents time and the Y-axis represents the voltage or processor speed. We assume that the voltage and frequency are exactly proportional. In the figure, the computation required by a task is represented by the 'area under the curve' for that task. Voltage or processor speed is normalized to the range [0...1]. The height of a task (voltage or processor speed) and the running time determine the energy required to complete the task. Assuming that three jobs (T_1 , T_2 and T_3) in Figure 1 (a) are executed with full voltage (1 volt for example) and the processor becomes shutdown when idle, the energy consumed for the three jobs is 8 energy units. With DVS, the supply voltage can be reduced to 3/5, 1/3 and 2/3 for the three jobs, respectively, which results in the same amount of computation (the same area under the curve) but they require less energy consumption of only 2.97 energy units² as shown in Figure 1(b).

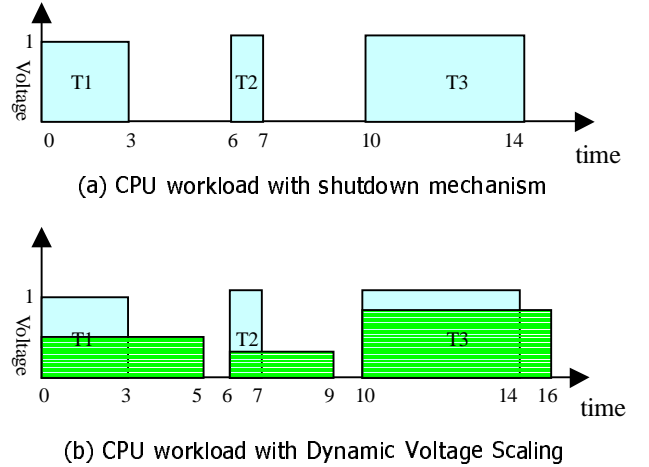


Figure 1: Voltage scheduling graphs by two mechanisms

DVS based on Previous Workload for Non-Realtime Systems

Since the dynamic voltage scaling relies on system workload, information on workload is critical to adapt

² The value is calculated from $(3/5)^3 \times 5 + (1/3)^3 \times 3 + (2/3)^3 \times 6 = 2.97$. Voltage values are triplicate (quadratic factor is for the supply voltage itself and one more factor is due to the clock frequency) and it is multiplied with the running time.

DVS efficiently. But it is not usually possible to know exact future workload of non-realtime systems. Instead, there are several prediction algorithms to heuristically estimate the workload of the near future. *Interval-based DVS* introduced in Section 1 is one such algorithm. Some complicated algorithms [5] estimate the future workload based on two parameters: `run_percent` and `excess_cycles`. `run_percent` is the fraction of cycles where the CPU is active in an interval. `excess_cycles` is the cycles left over from the previous interval spilled over into later intervals when speed is not fast enough to complete an interval's work. Five representative DVS algorithms are *PAST*, *FLAT*, *LONG_SHORT*, *AGED_AVERAGES* and *CYCLE* [5,6]. These algorithms scales the supply voltage based on workload prediction but they are different in the way the two aforementioned parameters are used.

DVS for Real-time Systems

In a real-time system, tasks are specified by $\{S_i, C_i, D_i\}$ where S_i indicates the task start time, C_i the computational resources required, and D_i the task deadline. Start times and deadlines are specified as absolute times and computational resources required are specified as execution time with the processor running at full speed. For real-time systems having hard deadline requirements, the voltage-clock scaling must be carried out under the constraint that no deadline is missed. Optimal voltage schedule is defined to be one for which all tasks complete on or before deadlines and the total energy consumed is minimized. Figure 2 depicts three tasks with $S_i = 0$ and differing deadlines. For such a schedule, with ordered deadlines $D_i \leq D_j \quad \forall i < j$, the optimal voltage or processor speed is P_k at time T , where k is the least index such that $T \leq D_k$, $P_k = \text{MAX}(W_l, l \geq k)$ and $W_l = \frac{1}{D_l} \sum_{m=1}^l C_m$ [11]. It requires

$O(n)$ time to schedule n tasks but extending this algorithm to the general case where $S_i \neq 0$ is non-trivial. Here, the intermediate workload, W_l , determines the optimal schedule for all tasks with deadlines at or before D_l . P_k calculates the processor speed to use such that future deadlines are not violated.

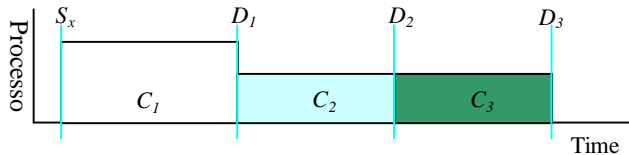


Figure 2: Optimal voltage scheduling with $S_i = 0$

Ishihara and Yasuura [8] considered both the

requirement of completing a set of tasks within a fixed interval and the number of switching activities for each task. The static voltage-scheduling problem is also proposed and formulated as an *integer linear programming problem*. In [9], a dynamic voltage-scheduling algorithm for real-time system was proposed. They defined an *occupation period* as the maximum time period that the next executed task can use without violation of real-time constraints for the future tasks. Now, the real-time constraints are always guaranteed if the next task is assigned a supply voltage so as to finish within the occupation period. The length of the occupation period is dynamically calculated and the voltage is set with respect to the occupation and the worst-case execution cycle of the task. For real-time systems with two voltage modes (*low mode* and *high mode*), static and dynamic voltage scaling methods were proposed in [10]. More recently, investigation has begun into *thread-based voltage schedulers* that require knowledge of individual thread deadlines and computation required [11]. With such information, thread-based schedulers calculate the optimal speed and voltage resulting in minimized energy consumption.

2.2. MPEG Decoding

This section discusses the basic structure of MPEG stream as well as decoding time and decoding strategy for real-time play. The MPEG video compression standard [15] defines a video stream as a sequence of still images or frames. A standard MPEG stream is composed of three types of compressed frames: *I*, *P* and *B*. While *I* frames are *intra-coded*, the generation of *P* and *B* frames involves, in addition to intra-coding, the use of *motion prediction* and *interpolation techniques*. As a result, *I* frames are, on the average, the largest in size, followed by *P* frames, and finally *B* frames. A *Group of Pictures (GOP)* is a sequence of frames from one *I* frame to the next *I* frame. Most encoders use a fixed *GOP pattern* when compressing a video sequence, where the *GOP pattern* specifies the number and temporal order of *P* and *B* frames between two successive *I* frames. Each video has different *GOP* sequences and numbers [16-17].

Another characteristic of a MPEG decoder is its realtime constraints. Frames should be passed with a given *frames per second (fps)* either played or dropped. Standard MPEG I provides eight values of *fps*: 23.976, 24, 25, 29.97, 30, 50, 59.94 and 60. With this constraint, a decoder should monitor the current system speed with real-time clock, and decide each frame either to drop or play. Decoders usually maintain four real-time phases, *DELAY-PHASE*, *B-PHASE*, *P-PHASE* and *I-PHASE* [18]. If the system is too fast to meet the real-time constraint, the decoder sets mode as *DELAY-PHASE* which delays between frames according to the delay

value. The delay value shows how fast the system speed is. On the contrary, if the system is unable to play all the frames, then some of the *B* frames are discarded (*B-PHASE*). If the system is still too slow even all the *B* frames are skipped, then some of the *P* frames are dropped (*P-PHASE*). Finally, *I* frames can be dropped in the worst case (*I-PHASE*). Since *I* frames and *P* frames are referred by *P* or *B* frames, dropping *I* or *P* frames causes other frames referring those to be dropped also. From decoding time point of view, *I* frames usually take longer than other frames while *B* frames take the least. On the contrary, *I*-frame decoding time per byte (*I_DTPB*) is the shortest while *B*-frame decoding time per byte (*B_DTPB*) is the longest. This is because a large computational work is needed in motion prediction in case of *B* and *P* frames. The frame decoding time can be predicted by frame type and size, and the corresponding predictor is shown to have less than 25% of prediction error [12]. In the next section, we exploit the predictor to develop our proposed DVS algorithm.

3. DVS Algorithms on MPEG Decoding

In this section, we present two algorithms that minimize energy consumption on MPEG decoding. The first algorithm is based on prediction by previous workload, while the second uses predicted MPEG decoding time as well as previous workload.

3.1. DVS with Delay and Drop Rate Minimizing Algorithm (DVS-DM)

Two important parameters that represent system workload in MPEG decoding are *delay* and *drop rate*. The delay value is set when the system is too fast to meet the realtime constraint. Large delay value means that the system is too fast. If the delay value is zero, the system speed is adequate to decode the scene or slow which causes dropping frames. Drop rate is the rate at which the decoder drops frames when it is not able to process all incoming frames. There are three kinds of drop rate: *B-frame drop rate*, *P-frame drop rate* and *I-frame drop rate* that are set on *B-PHASE*, *P-PHASE* and *I-PHASE*, respectively.

DVS with Delay and Drop Rate Minimizing Algorithm (DVS-DM) tries to minimize both delay value and drop rates. At the beginning of a GOP, the supply voltage is scaled according to the delay value or drop rate. If the delay value is positive (*DELAY-PHASE*), voltage and frequency is scaled down in proportional to the delay value. Otherwise, if the system is in *B-PHASE*, voltage is raised in proportion to the current drop rate. If the system is in *P-PHASE* or *I-PHASE*, the voltage is set to the maximum voltage regardless of the drop rate. The C code which implements this algorithm is shown in Figure 3.

```
static void scale_voltage_delay( )
{
    /*voltage is set to maximum voltage*/
    if((phase==I_PHASE)|| (phase ==P_PHASE))
        set_max_voltage( );
    else if(phase == B_PHASE)
    {
        /* scale up voltage */
        current_voltage = current_voltage
            + drop_rate*increase_factor;
        set_voltage(current_voltage);
    }
    else if(phase == DELAY_PHASE)
    {
        if(delay<100)/*keep current voltage*/
            return;
        else { /* scale down voltage */
            current_voltage =current_voltage
                - delay*decrease_factor;
            set_voltage(current_voltage);
        }
    }
}
```

Figure 3: DVS with delay and drop rate minimizing algorithm (DVS-DM)

3.2. DVS with Predicted MPEG Decoding Time (DVS-PD)

DVS-DM is based on the workload history. Voltage is scaled according to the variables (delay and drop rate) which are set by previous workload. It may not work efficiently because the workload fluctuation of MPEG decoding is usually high as will be explained with actual movie streams in Section 4.1. The high fluctuation is due in part to the fact that a given MPEG video stream contains different frame types, and in part to the potential wide variation between scenes (e.g., talking heads versus action). This makes the prediction of decoding time based on the past behavior difficult. We propose an efficient DVS algorithm which uses both the previous history and the estimated decoding time, called *DVS with Predicted MPEG Decoding Time (DVS-PD)*.

Decoding time is estimated at the beginning of each GOP by considering MPEG frame types and sizes as well as previous workload information. As explained in Section 2.2, when decoding a stream, each frame type has different decoding time per byte (DTPB). DTPBs for *I*, *P* and *B* frames are not the same, and in addition these values differ by scene characteristics and vary throughout the MPEG decoding. To adapt to workload changes and scene characteristics, DVS-PD first maintains DTPBs by weighted sum of previous average of DTPBs and the last DTPBs in `update_statistics` routine as shown in Figure 4. The

weight_factor controls the relative weight of the most recent and past history of DTPBs. This number is set to a specific number (0.4 in our simulation with which the results are the best). But, as the best weight_factor is stream specific, no optimal weight_factor for all streams exists. We then get *I*, *P* and *B*-frame sizes of next GOP and estimate the next GOP decoding time by multiplying frame sizes and DTBP values. Finally the voltage and frequency for the next GOP are set according to the estimated GOP decoding time.

balance_factor is introduced to measure the gap between the predicted workload and real workload of the recent interval and it is determined by delay and drop rates as well as three constants, I_P_balance, B_balance and DELAY_balance. If the balance_factor is below zero, it means that the predictor under-estimated the workload of the previous interval causing frames to be dropped. If the balance_factor is positive, the prediction causes delay by overestimating workload. Therefore, the prediction is corrected by adding balance_factor to the estimated voltage, which helps algorithm to be more adaptively coping with workload changes. This algorithm is shown in Figure 4.

```
static void scale_voltage_predict( )
{
    update_statistics( );
    get_GOP_size( );
    decode_time= est_decode_time( );
    volt=set_volt_predict( decode_time );
    set_voltage(volt);
}
void update_statistics( )
{
    /* weighted average */
    avg_I_DTPB=avg_I_DTPB*(1-weight_factor)
                + I_DTPB *weight_factor;
    avg_P_DTPB=avg_P_DTPB*(1-weight_factor)
                +P_DTPB*weight_factor;
    avg_B_DTPB=avg_B_DTPB*(1-weight_factor)
                + B_DTPB * weight_factor;
}
void get_GOP_size( )
{
    I_size = get_GOP_I_size( );
    P_size = get_GOP_P_size( );
    B_size = get_GOP_B_size( );
}
double est_decode_time( )
{
    I_decode_time = I_size * avg_I_DTPB;
    P_decode_time = P_size * avg_P_DTPB;
    B_decode_time = B_size * avg_B_DTPB;
    return I_decode_time + P_decode_time
```

```
        + B_decode_time;
    }
void set_volt_predict( d_time )
{
    if((phase==I_PHASE)|| (phase==P_PHASE))
        balance_factor = I_P_balance;
    else if(phase == B_PHASE)
        balance_factor = drop_rate*B_balance;
    else if(phase == DELAY_PHASE){
        balance_factor = delay*DELAY_balance;
    /*scaled well, thus no volt. change occurs*/
    if(delay<100)
        {
        /*standard voltage and time are set to
        current values to adjust to current
        workload*/
            std_voltage = voltage;
            std_dtime = d_time;
        }
    return std_voltage*d_time/std_dtime
        + balance_factor;
}
```

Figure 4: DVS with predicted MPEG decoding time

4. Performance Evaluation

4.1. Experimental Environment

In this section, we briefly overview the MPEG decoders as well as the video streams used in our experimental study. We also present the assumptions we made for simplifying the model.

MPEG Decoders

With six sample streams explained later in this section, we compare four MPEG decoders. The first one is a modified MPEG decoder from Boston University [19]. This decoder is based on Berkeley MPEG decoder but added the functions for real-time and multi-stream play. To support the realtime play, the player dynamically adapts or drops frames to match the number of frames passed through the player to the frame rate given in the sequence header of the video stream. The second one is the MPEG decoder with the shutdown algorithm explained in Section 2.1, the third one implements *DVS-DM*, and the last one runs *DVS-PD*. Though the first, the third and the fourth MPEG decoders are implemented and executed to measure the consumed energy, the second MPEG decoder with the shutdown algorithm is not a real implementation. We calculated the energy consumption based on the values of the original

MPEG decoder assuming that the shutdown occurs immediately when the system is idle, and no energy is consumed on shutdown and wake-up. As in the equation below, the delay time is removed out of the whole decoding time of the original MPEG decoder.

$$E_{shutdown} = E_{original} \times \frac{Decode_Time - Delay_Time}{Decode_Time}$$

$E_{shutdown}$ and $E_{original}$ are the energy consumptions by the shutdown algorithm and the original MPEG decoder, respectively. $Decode_Time$ is the total decode time by the original MPEG decoder and $Delay_Time$ is the total time period during which the decoding is delayed in DELAY_PHASE explained in Section 2.2. This is reasonable because the original decoder busy waits when the system should be delayed to meet the real-time constraints while the decoder with shutdown mechanism shuts down the processor instead of busy waiting.

Table 1: Sample video clips used in the experimental study

Movies	Avg. GOP Size	Frames Per Second	Avg. Dec. Time	Dev. of GOP Dec. Time	Fluct.
Alien	96066	23.976	0.378	0.00920	0.0000745
Conan	76778	30	0.232	0.00057	0.000007
The Close Shave (Shave)	96065	23.976	0.531	0.00115	0.000025
X-men	76775	30	0.272	0.00029	0.000001
Hollow Man (Hollow)	96066	23.976	0.313	0.00094	0.000013
Charlie's Angel (Angel)	92131	25	0.267	0.00834	0.0000565

MPEG Streams

We selected six famous movies including two animations for evaluating our decoders with respect to energy consumption. Instead of playing the whole movie, we used a portion (5000 frames) from each movie. We characterize the six movies with respect to average frame size, average decoding time, deviation of GOP decoding time and fluctuation of frame decoding time as shown in Table 1. Fluctuation($fluct$) is a degree of differences between adjacent GOP decoding times as defined below, where $decode_time_i$ is the GOP decoding time of the i -th GOP and num_GOP is the number of GOP's in the stream.

$$fluct = \frac{\sum (decode_time_i - decode_time_{i-1})^2}{num_GOP}$$

High fluctuation leads to high errors in predicting the GOP decoding time based on previous history. Action movies, such as *Alien* and *Charlie's Angel* in the sample streams, usually have large fluctuations.

Assumptions

We made the following assumptions for our experimental study. Firstly, it is difficult to construct a real DVS system consisting of frequency scaler running on the fly, efficient DC to DC converter which convert voltage with small latency, and additional circuits to adjust and sink the frequency and voltage [10]. In our experiment, we emulate a real DVS system by scaling the frame rate. Every stream has a fixed number of frames per second. If we double the rate, the CPU is busy twice, which is equivalent to reducing the operating frequency in half. In order to hold the idea, we need to assume that there is no memory operation on MPEG decoding because frame-rate scaling does not scale the memory access time. This assumption is valid as memory operation time in MPEG decoding is relatively small compared to the CPU operation time [20].

Secondly, we exclude the display function of MPEG players because the state-of-the-art technology executes those routines separately on other hardware devices. Only *IDCT* and *reconstruction* are considered in our experiment, which are the main part of the decoder, and *parsing* and other minor operations are ignored as they take less than 3 percent of total decoding time. The third assumption is that the machine was considered to use no energy when idle, which is a bit optimistic because a chip will draw a small amount of power while in standby mode. The fourth assumption is that it takes no time to switch speeds. In practice, raising the speed will require a delay to wait for the voltage to rise first, although we speculate that the delay is on the order of 10s of instructions. Finally, we assume that the voltage is exactly proportional to the operating frequency. This is a reasonable assumption except when the voltage approaches the threshold voltage.

4.2. Simulation Results

In this section, we present the experiment results. We first show the performance benefit by employing DVS-PD in terms of energy consumption and then present the performance loss in terms of quality of service. Figure 5 shows the relative energy consumption by four MPEG decoders with six sample streams assuming that the original MPEG decoder consumes one unit of energy. It is observed that DVS-PD consumes the least energy among the four

decoders. Depending on the streams, it saves 13%-56% compared to the conventional shutdown algorithm. DVS-DM is also better than the shutdown algorithm except *Charlie's Angel*. The exception occurs partly due to the high fluctuation. Since DVS-DM predicts the future workload depending on previous workload only, a movie stream with high fluctuation makes it difficult to predict accurately. As described in Section 4.1, the energy consumption with the shutdown algorithm is not measured but calculated by the GOP decoding time and delay time. Thus, the energy values are approximately proportional to the average GOP decoding time shown in Table 1. Energy consumption due to the wake up time is not considered, but it may not be ignorable in real systems as it takes not only energy but also time.

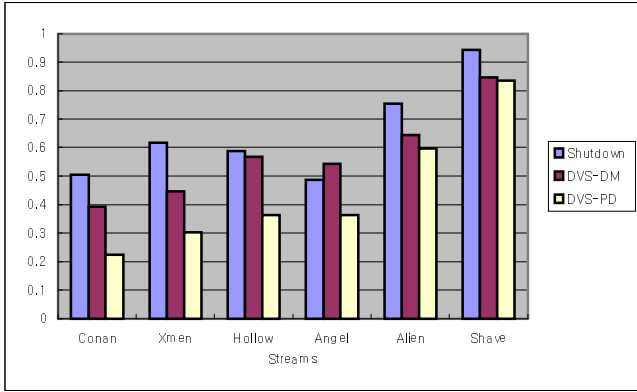


Figure 5: Relative energy consumption
(Assume that the original MPEG decoder consumes one unit of energy.)

The main reason behind the successful energy saving with DVS-PD is that the next workload is predicted more accurately than other approaches. We measured the prediction accuracy or inaccuracy in terms of *error rate* by the following equation.

$$err_rate = \frac{\sqrt{\sum (est_dec_time - dec_time)^2}}{num_GOP}$$

where *est_dec_time* is the estimated GOP decoding time, *dec_time* is the real GOP decoding time and *num_GOP* is the number of GOP's. Error rates of the six sample streams are less than 10% as shown in Figure 6. Notice that the error rate is not related with fluctuation of the movie streams, which is clearly shown in the figure where the streams are displayed with the ascending order of the degree of fluctuation in Table 1. In other words, DVS-PD performs well even with streams with high fluctuation. However, the error rate strongly affects the performance improvement of DVS-PD over the shutdown scheme in terms of energy consumption. As shown in Figure 5, the DVS-PD saves

energy more than 50% with X-men but it becomes as small as 15% with Shave. This is expected according to Figure 6, where the energy rate of X-men is the least and that of Shave is the largest. We can, therefore, conclude that more accurate prediction results in more energy saving.

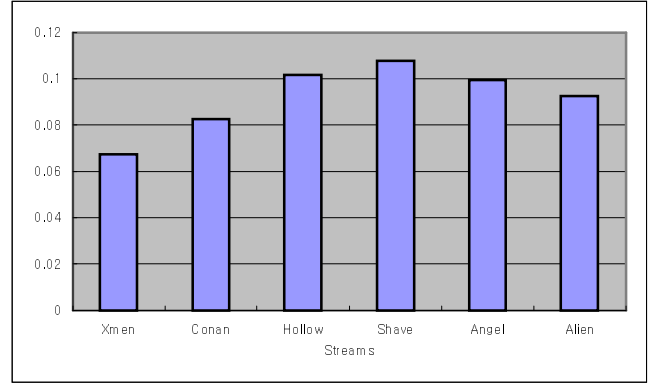


Figure 6: Prediction inaccuracy or error rates with DVS-PD
(Streams are ordered with ascending degree of fluctuation.)

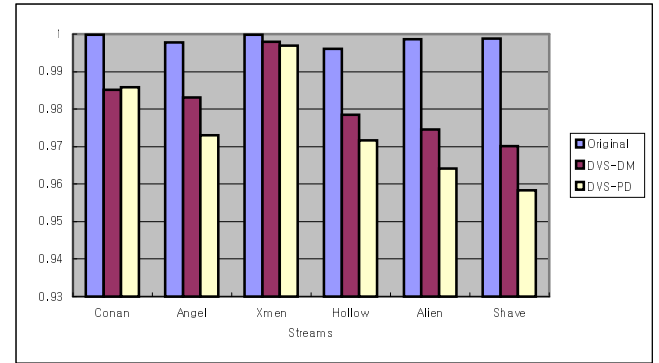


Figure 7: QoS or ratio of the number of played frames to that of passed frames
(Streams are shown with ascending order of average GOP decoding time.)

While DVS-PD improves energy efficiency of MPRG decoding, it may in return degrade the quality of service (QoS) in playing a real-time MPEG stream. Figure 7 shows the experimental results of the number of played frames to the number of passed frames, which represents the QoS of MPEG decoding. DVS-PD performs the worst, which means more frames are dropped during the MPRG play. However, it is noted that QoS values are still more than 95% and thus, the quality loss is ignorable. Overall QoS with DVS-DM is better than DVS-PD. This is due to the inherent characteristic of DVS-DM algorithm that tries to minimize drop rate. In Figure 7, six streams are ordered with average GOP decoding time in Table 1 and we found that a movie stream with longer decoding time has worse QoS parameter with DVS-DM and DVS-PD except the case of *X-Men*.

5. Conclusions

Dynamic voltage scaling is a powerful methodology for reducing power consumption of a processor. But applying this method to MPEG decoding is not straightforward mainly due to variability of the workload of MPEG decoding. In this paper, we introduced two DVS algorithms on MPEG decoding: *DVS-DM (DVS with delay and drop rate minimizing algorithm)* and *DVS-PM (DVS with prediction of MPEG decoding time)*. They save energy by scaling the supply voltage based on the prediction with previous workload and predicted MPEG decoding time. We compare four MPEG decoders with six sample streams and found that DVS-PD shows the best performance with respect to energy consumption and DVS-DM is slightly better than the conventional shutdown algorithm. Outstanding energy saving with DVS-PD is due to higher prediction accuracy of future workload than other approaches. It is also found that energy saving is closely related with average decoding time and fluctuation. With DVS-DM, high fluctuation makes it difficult to predict future workload based on the previous workload only and it results in low efficiency. On the contrary, it is found that DVS-PD is not much affected by the fluctuation. Instead, performance of DVS-PD in terms of energy consumption depends on the error rate of the predictor, which implies that if decoding time is predicted more accurately, DVS algorithm can be more efficient.

References

- [1] T. Burd and R. Brodersen, "Energy Efficient CMOS Microprocessor Design," *28th Hawaii Int'l Conf. on System Science*, Vol. 1, pp. 288-297, Jan. 1995.
- [2] T. Burd and R. Brodersen, "Design Issues for Dynamic Voltage Scaling," *Int'l Symp. on Low power Electronics and Design*, pp. 9 – 14, 2000.
- [3] W. Namgoong, M. Yu, and T. Meng, "A High-Efficiency Variable-Voltage CMOS Dynamic DC-DC Switching Regulator," *IEEE Int'l Solid-State Circuits Conf.*, pp. 380-381, 1997.
- [4] K. Suzuki, et al., "A 300 MIPS/W RISC Core Processor with Variable Supply-Voltage Scheme in Variable Threshold-Voltage CMOS," *IEEE Custom Integrated Circuits Conf.*, pp. 587-590, 1997.
- [5] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy," *First Symp. on Operating System Design and Implementation OSDI '94*, pp. 13-23, Nov. 1994.
- [6] K. Govil, E. Chan, and H. Wasserman, "Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU," *Technical Report TR-95-017, ICSI Berkeley*, Apr. 1995.
- [7] T. Pering, T. Burd, and R. Brodersen, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms," *Int'l Symp. on Low Power Electronics and Design*, Aug. 1998.
- [8] T. Ishihara and H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," *Int'l Symp. on Low Power Electronics and Design (ISPLED '98)*, pp. 197-202, Aug. 1998.
- [9] T. Okuma, T. Ishihara, and H. Yasuura, "Real-Time Task Scheduling for a Variable Voltage Processor," *12th Int'l Symp. on System Synthesis*, pp. 24-29, 1999.
- [10] Y. H. Lee, C. M. Krishna, "Voltage-Clock Scaling for Low Energy Consumption in Real-time Embedded Systems," *Sixth Int'l Conf. on Real-Time Computing System and Applications*, pp. 272-279, 1999.
- [11] T. Pering, R. Brodersen, "Energy Efficient Voltage Scheduling for Real-Time Operating Systems" *IEEE Real-Time Technology and Applications Symp., Works In Progress Session*, 1998.
- [12] A. Bavier, B. Montz, and L. Perterson, "Predicting MPEG Execution Times," *SIGMETRICS/ PERFORMANCE '98, Int'l Conf. on Measurement and Modeling of Computer Systems*, pp. 131-140, Jun. 1998.
- [13] LongRun technology from Transmeta
<http://www.transmeta.com/crusoe/lowpower/longrun.html>
- [14] R. Golding, P. Bosh and J. Wilkes, "Idleness is not Sloth," *HP Laboratories Technical Report HPL-96-140*, 1996.
- [15] J. Mitchell, W. Pennebaker, *MPEG Video Compression Standard*, Chapman & Hall, 1996.
- [16] M. Krunz and S. Tripathi, "On the characterization of VBR MPEG streams," *ACM SIGMETRICS, Int'l Conf. on Measurement and Modeling of Computer Systems*, pp. 192-292, 1997.
- [17] W. Lou, L. Chia and B. Lee, "Characterization and Source Modeling of MPEG-2 VBR Video Source," *ICICS, Int'l Conf. on Information, Communications and Signal Processing*, Vol. 3, pp. 1652 -1656, 1997.
- [18] J. A Boucher, Z. Yaar, E. J. Rubin, J. D. Palmer, and T. D. C. Little, "Design and Performance of a Multi-Stream MPEG-1 System Layer Encoder/Player," MCL Technical Report 01-07-1995, *IS&T/SPIE Symp. on Electronic Image Science & Technology, San Jose, CA*, Feb. 1995.
- [19] MPEG Decoder from Boston University
<http://hulk.bu.edu/pubs/software.html#system-player>
- [20] K. Patel, B. Smith and L. Rowe, "Performance of a Software MPEG Video Decoder," *First ACM Int'l Conf. on Multimedia*, pp. 75 – 82, 1993.